UNIVERSITY OF OSLO
Department of Informatics

The potential of Trusted Computing for Strengthening Security in Massively Multiplayer Online Games

Master thesis

Øyvind Skaar

May 3, 2010

Abstract

Massively Multiplayer Online Games have become a popular leisure activity and big business. They have also become targets for attacks beyond the personally motivated cheating attempts. The opportunity to make real money by exploiting these virtual worlds have made security a higher priority. Combating the security problems of multiplayer games in software alone is doomed to fail, because the attacker is the player which already controls the system.

In this thesis, I investigate security problems in massively mulitplayer online games, and how the technologies collectively known as Trusted Computing can be used to combat these problems. To this end, I have created a new taxonomy, attack-trees and an attack-graph to categories and detail current security problems. These resources are then used when investigating the different Trusted Computing technologies. Practical problems with the use these technologies are identified, and what security problems they can solve is discussed.

The conclusion is that the classic trusted computing technologies would solve the major issues, but require changes to common architecture and can therefore not be implemented by the game developers alone. The newer, dynamic root of trust based, technologies have great potential as long as the hardware becomes pervasive and a few underlying problems are solved.

Preface

This thesis was written as a part of my masters degree in computer science at the Department of Informatics (ifi), University of Oslo, and the University Graduate Center (UNIK).

This work initially required much research into the security problems in online games, as the project was somewhat undefined. The initial goal was to solve at least some of these problems with a novel idea. After research into the problems affecting online multiplayer games, current and novel solutions, I returned to one of the original ideas and dove into the chaos that is trusted computing.

I would like to take this opportunity thank my advisors — Audun Jøssang at UNIK and Jarle Snertingdalen at Funcom — for much-needed feedback and guidance.

Øyvind Skaar
May 3, 2010, Oslo

Contents

List of Figures

Chapter 1

Introduction

Millions of people currently participate in persistent, online virtual worlds created by Massivly Multiplayer Online Games (MMOs). These games are similar to other multiplayer games in that they let players interact, compete and cooperate with each other. But they distinguish themselves with a huge scale and a persistent virtual world. Some of these games have been hugely successful in the last decade. Most notably World of Warcraft (WoW) from Blizzard Entertainment, with, according to themselves, 11.5 million subscribers in late 2008 [1]. Although it should be noted that these numbers reflect accounts and not necessarily active players, WoW undoubtedly has a large user base. Most of these games have virtual economies witch their own virtual currency. The fact that there are ways to convert virtual currency and items into real money have helped to make hacking these games not just a hobby, but very a profitable, yet low risk, activity. Also, games and game development are often on the cutting edge of technological development and lessons learned from games security can potentially be applied to other areas of software as well.

Currently, security problems in these games are recurring and widespread. There seems to have been a recent increase in studies related to games security. This is likely due to the increasing popularity of games, and that security problems have become noticeable to users. Games share most of the security problems of other types of software and have in addition several game specific security problems. This includes cheating, which is a serious problem for many reasons. It can discourage honest players from participating in a specific game, and potentially make them switch to competing games. Also, if becoming too widespread, cheating can ruin the virtual worlds, especially their economies, and cause loss of reputation and revenue. While cheating is specific to games, it is not surprising that MMOs, being distributed net-

worked applications, also face more conventional security problems. Just as cheating, these problems such as information leakage or DOS attacks, can be means to an end or the end itself. Traditionally, getting an edge in games and just curious exploration were the main motivations for breaking the security of games. But with the rise of MMOs, real money can be made through hacking and tampering with online games [1]. In response to the security threats, developers have taken, sometimes controversial, technical and legal countermeasures.

Trusted Computing is a class of technologies developed and promoted by the Trusted Computing Group. One of their goals is to protect the integrity of client software. The Trusted Platform Module (TPM) was their initial specification and proposal to strengthen the security of personal computers. Somewhat controversial, this technology is especially interesting for game security because it, in a sense, puts some restrictions on the user of the system.

This thesis centers around MMOs, specially the client-side where most of the current problems lie. Although the TPM, and Trusted Computing in general, are purposed as a remedy for cheating and other game security issues ([2, 3, 4, 5]), there is a lack of investigation into how this should be done. The goal of this thesis is to determine what, if any, of the technical security problems facing MMOs can be solved with Trusted Computing. To do this, a taxonomy of security problems, as well as attack-trees, are created, and practical use of Trusted Computing technology is investigated.

A brief history and an overview of multiplayer computer games, what they are and how they work, is given in chapter 2. Chapter 3 analyses security problems in MMOs, purposes a new taxonomy and attack-trees based on common motivations. Trusted Computing is explained in chapter 4, and a possible solutions are discussed. Chapter 5 summarises this thesis and recommendations for future work are given.

[1] The same is true for online betting and gambling

Chapter 2

Background and Motivation

Computer and video games have come a long way since their humble beginnings more than fifty years ago. According to the Entertainment Software Association, computer and video game sales grew to U.S. $11.7 billion in 2008. The numbers are for the U.S. alone, and, presumably, only includes games sold in stores and not online downloads. They also find that 60% of American households play these games.[6]

There are many categories of electronic games: computer games, video console games, arcade games and games played on hand held devices. The most advanced games can arguably be found in the computer games and video games categories. Video (console) games are played on special build video game consoles typically connected to a TV, while computer games are played on (more or less) standard personal computers.

This thesis focuses on computer games, but there are many similarities, and the same game is often released for both the PC[1] and one or more of the video game consoles such as the Microsoft Xbox 360 and the Sony Playstation 3.

2.1 Multiplayer Games

The traditional form of video gaming has one player interacting with the computer or gaming-system, often trying to "beat" preprogrammed challenges, as seen with the arcade games made popular in the late 70's, and later console and pc-games. Games also often consist of "AI"[2] controlled opponents, that is, characters in the game that are partly pre-programmed and partly

[1] Typically Microsoft Windows, but there exists games for other operation systems as well.

[2] Artificial Intelligence

Figure 2.1: Multiplayer games: Past (British Legends) and present (World of Warcraft). From [7]

respond and react "intelligently". This form of gaming is often called "single player" because the gamer does not play with or against other humans, but is engaged in the game alone.

A more advanced and complex form of games are multiplayer games where the player is interacting with other players in some way, often in (near) real time. Multiplayer existed long before the Internet. Arcade games and game consoles have long supported two or more players, e.g., by dividing ("splitting") the screen or connecting several systems with a dedicated cable. Computer games ran at supercomputers supported several players who connected using terminals.

But the commonplace of Internet connectivity has contributed to making multiplayer games more popular by providing the technological basis for several types of new multiplayer schemes and new business models for games. Suddenly it was possible to connect and interact with players "everywhere". Although other types of multiplayer games still exist, utilizing the Internet is the most common way to play with others.

2.2 Massively Multiplayer Online Games

Although there are many different types of multiplayer games, this thesis focuses on so-called "massively multiplayer online games" (MMOs[3]).

[3]or MMOGs

"Massively multiplayer" means that there is not just a couple, but hundreds, thousands or tens of thousands of players playing together.

One inspiration of the modern MMOs seem to have been the text-only multiplayer games developed in the late 70's, called Multi-User Dungeons (MUDs). These games were small in scale compared to modern MMOs, but had many of the same traits and operated in a somewhat similar manner. A MUD typically consisted of a virtual world with many rooms. The player would connect using a text-only client and get a description of the in-game surroundings including other players, objects, monsters and non-playable characters (NPCs). Actions, such as fighting and moving around, could be performed by typing commands. [7].

Modern MMOs have an entirely different user interface (figure 2.1) and support more concurrent players, but many of the game mechanics are similar.

Persistent Worlds The main feature of an MMO, besides the scale, is the ongoing virtual world where users can create virtual, customizable characters. These character can resemble the user themselves or something else entirely[4]. The virtual and artificial world continues to exist and develop also after the player has disconnected and is no longer playing his character. The players who are online inhabit this world and, together with computer-controlled characters (NPCs) and events, change its state. Players are exploring, fighting and socializing, and since the world is persistent, their achievements the one day are still there the next.

Role Playing Games Currently the most popular MMOs are role playing games, and thus abbreviated MMORPG. These games are inspired by traditional role-playing games like Dungeons & Dragons, and let the player assume a role, a fictional character in the game. This character can develop as the game progresses, gaining experience, wealth, items, magic spells and other skills. Building, or leveling up, your character in games like these often takes considerable time and effort.

Through these kinds of games people can be entertained, and can play roles that would be impossible to take on in real life.

Some of these games have been hugely successful in the last decade. Most notably World of Warcraft (WoW) from Blizzard Entertainment, with, according to themselves, 11.5 million subscribers in late 2008[1]. Although it

[4]There are, for example, no reason your character can't be another gender or another species

should be noted that these number reflect accounts and not necessarily active players, WoW undoubtedly has a large user base.

Virtual Economies Most MMOs have virtual economies witch their own virtual currency. There are ways to convert virtual currency and items into real money. If a player wants in-game achievements, such as items, virtual money, or a character with better abilities, they can buy these from other players instead of investing the substantial time and effort necessary to gain them in the game. This has created a market where virtual items and characters for popular MMOs are bought and sold with real (non-virtual) currency, mostly US dollars or Euros. Some developers, such as Blizzard, deem such trade a violation of the EULA[5], but it is widespread nonetheless with the help of third parties. Other developers support this trade with their own virtual currency exchange, while yet others have this as the game's main source of income.

2.3 MMO System Architecture and Operation

2.3.1 The Client/Server Architecture

A MUD was typically run on a central server and the players would connect using modems or a local network. The MUDs were very small by todays standards, but their architecture has a lot in common with the ones most used today. Although some peer-to-peer games exist, and this technology is being researched heavily, the client/server model is still used by the vast majority of MMOs[7].

In theory the game itself could run on the server, with just the display and input/output redirected to the client. But in reality most modern games are resource-intensive and the servers are not powerful enough to run enough instances of the game. Also, current networks are not fast and reliable enough to transfer the required amount data in real-time.

The players run a *game-client* on their computers, where, generally, all static information such as game-sounds, graphics and other game data is stored and loaded from. The game-client connects to the server and continuously sends and receives information. Most actions performed by a player affect other players as well. This includes even simple things such as moving around in the world since the other players has to see your character moving. The game-client therefore sends information about the players actions to the game-server and receives similar information about the other players, as well

[5]End-user license agreement, typically agreed upon by the user before using the software

as other events in the game world. The server is responsible for redistributing all information to those game-clients who need it. This means that all traffic flows through, and therefore is controlled by, the game-server.

One thing to note is that most of what the player sees on her screen is rendered by the game on her local computer and has not traveled the network.

2.3.2 Handling a Massive Amount of Players

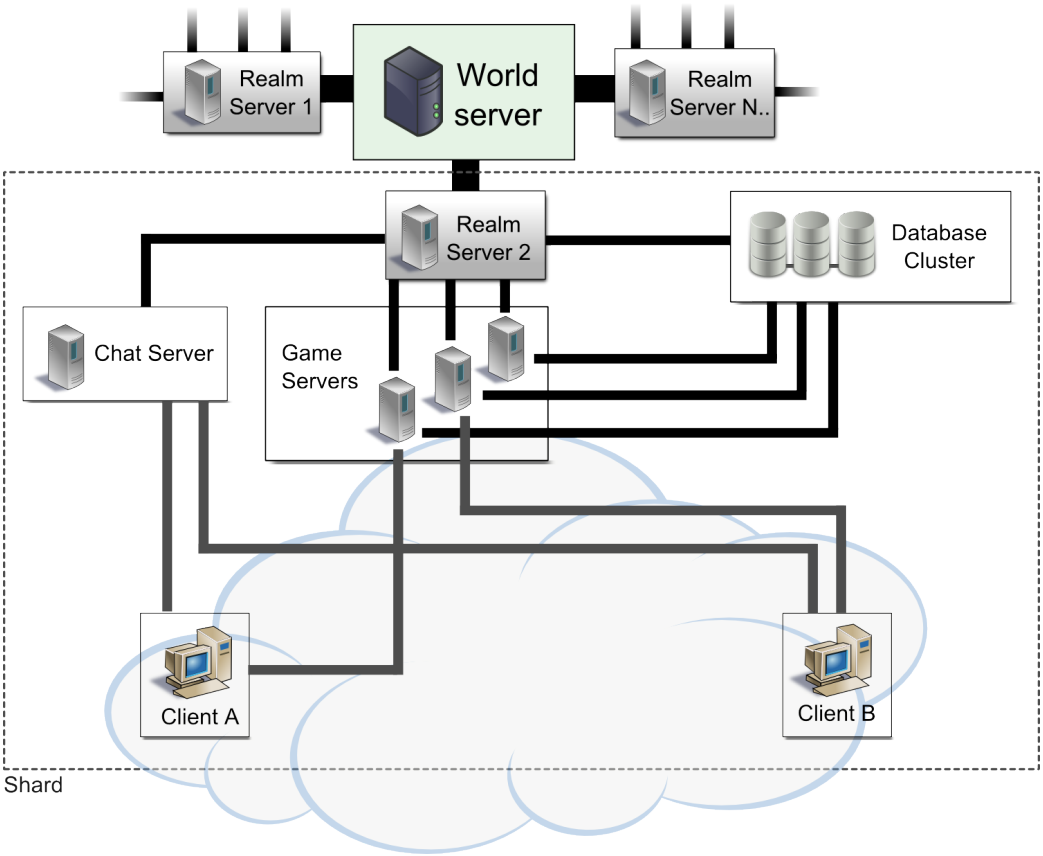

Figure 2.2: Simplified example MMO architecture. Client A and B connect to game servers through the Internet. On the server-side incoming connections are load-balanced to all game servers.

The old MUDs were small by todays standards. But they also had to cope with limited resources, and keeping information about all the players in the game in memory and exchanging this information proved challenging. The solution was to split the world into rooms. There where many rooms

and only those players who where in the same room could affect the room and each other. This way, each client only has to know and care about the other players in his current room.

In newer MMOs this concept still applies and is taken a step or two further. Splitting the world into many "rooms" or areas, and placing each area on a different server is a method used by the game *Everquest* as a way to distribute players and hence resource requirements. When a player travels to a different area the game-client disconnects from the current server and connects to the new one. When a player re-joins the game after leaving, the game server(s) or a separate log-in server remembers the last area the player visited and tells the game-client what server to connect to [8].

Sharding is another technique where the whole game world is duplicated and each player is bound to one of the "copies". This has the advantage that it's technically easy to scale up: just add a few servers and populate them with new players. The downside is that this can be "bad for business" when friends can not find each other because they are in different copies of the same game world. It can also be hard to strike a balance between too many and too few players, as the world has to have enough players to not seem deserted.

Figure 2.2 shows a simplified example MMO client/server architecture. Each realm server represents a shard, and each shard has a separate database cluster, a chat server and several game servers. The clients in Fig. 2.2 are connected to the same realm and therefore their characters are part of the same shard. Being part of the same shard, the clients characters are part of the same virtual world. Players can not easily move their characters to another shard or interact with players who are connected to different realms, and thus in other shards. Each client connects to one of the game servers and to a chat server. In this setup, actions by Client A updates the game server he is connected to, which in turn updates the databases and the realm server when appropriate. All this has to happen before those actions are visible to Client B. [8]

2.3.3 Out of Sync Game State

Unfortunately the Internet is not a perfect and instant communication channel. By the time the updates from one client reaches the other clients, the information will already be outdated.

In a perfect world the game-client would send an update for every event, but in practice bandwidth is limited, and updates will only be sent a fixed number of times pr. second.

These properties leads to clients who are out of sync; they do not have

the same view of the game world at any given time. When a central server is used it is often authoritative in the sense that whatever view it has on the game world and state is the correct one, and the clients have to adjust.

As long as the delay, or *lag*, in communication is small the game will provide a suitable workaround.

To counter the problem with infrequent updates, the game-client will make educated guesses as to where someone or something is located at any given time. It will look at the last update and try to calculate what has happened since then. Unfortunately this can be abused, for example, by purposely delaying outgoing updates. The lack of authoritative updates will force the others to guess more, and the cheater can wait to the others have made their move and then react accordingly [9]. In a client/server setup the server can be the authority.

When packets, and thus commands, arrive delayed or out-of-order, the server can look at a time-stamp in the packets to determine the real order of the actions. This, however, requires a global clock and trust in the game-client to time-stamp its packages correctly, i.e., not lie.

2.4 Why Games?

2.4.1 Big Business, Real Money

As the popularity of massively multiplayer games has risen the last decades and gaming has become a significant industry, security has become more and more important. With so much effort and money invested in developing and maintaining games the developer has much at stake. And the users, many spending countless hours building their characters in these games, and often a lot of money too, suffer if what they have worked so hard for is lost.

It seems like the issue of illegal copying often get more attention than security, but security flaws can ruin the game in many ways.

Because online games can represent value they also provide a greater incentive for players to cheat and for attackers and criminals to exploit games for profit.

The security firm Eset note that *"While there have always been unpleasant people who will steal another gamer's credentials just for the heck of it, trading in virtual cash, treasure, avatars and so on is now a major source of illegal income for cybercriminals"* [10].

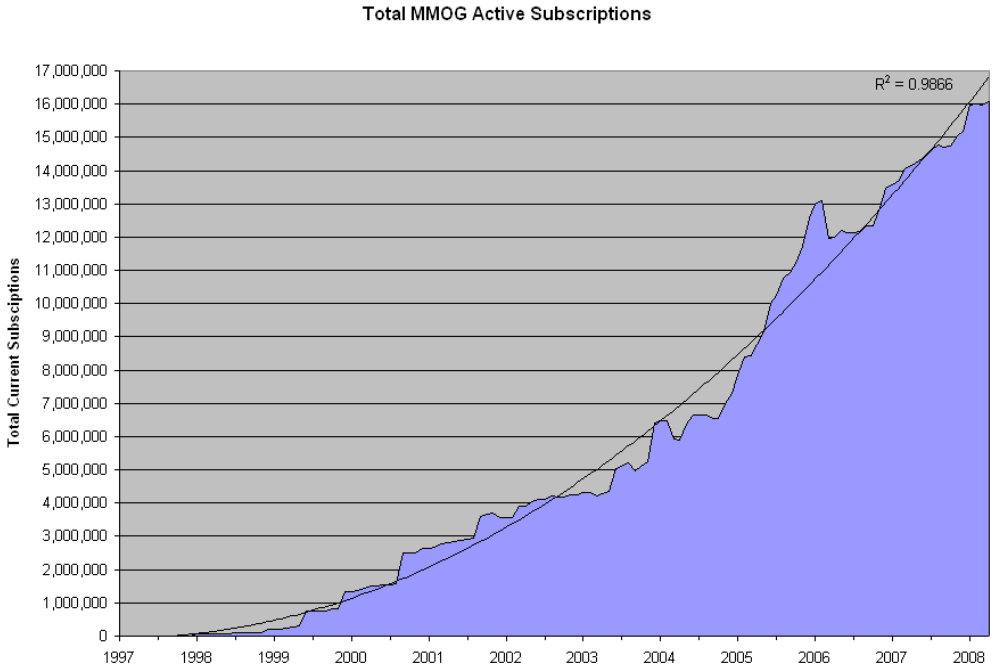

Figure 2.3: The total number of active subscriptions in MMOs. From mmochart.com (`http://www.mmogchart.com/Chart4.html`)

2.4.2 Complex and Distributed

For many years computer games have been some of the most complex and demanding programs to be run on personal computers, always demanding the newest and greatest equipment to deliver the best experience.

A culture where new "features" and better graphics are more important than stability and bug-free programs have helped the developers push the limits of what is technically possible. Multiplayer games have the added complexity of sharing a virtual world between all the players in real time.

The type of online games this thesis focuses on is characterised by being massively distributed. This, of course, does not make them any simpler. The "state" of the virtual world is everything that is dynamically changing. Data about in-game characters, player controlled and those controlled by the game itself (NPCs), is an example of information that depends on the actions of the players. All the data that do not change, such as graphics and sound data, are not shared in real time. The dynamic game world state however, has to be shared between thousands of players. Each game-client does not have to be informed about everything that goes on in the world, only the

pieces that affect it.

2.4.3 A Look Into the Future?

This is a type of massively distributed system not seen in too many other "real life" systems today. Not many other systems can make the claim that they connect millions of people together in complex, real time environments. The massively distributed model might become more popular in the future. Therefore MMOs can be a testing ground: what we learn here (what works/does not work, mistakes etc.) can be deployed in future systems.

Many people believe the future of computing will be more distributed, and right now there is nothing more distributed than MMOs.

2.4.4 New Threat

Game security is a fairly new topic. Although games and online games have been around for some time, the security of the games was for a while a problem only concerning the players and the developers.

Now that games have become more popular and even more complex, the topic has gained some academic interest, but it can still be seen as a new topic with a lot of unexplored territory and research to be done.

Chapter 3

Security Problems in Games

The security problems related to MMOs, and games in general, are many and diverse. This chapter gives a state of the art overview of these problems based on current literature, as well as attack-trees and a taxonomy.

3.1 Reasons to Cheat or Hack Multiplayer Games

Peoples' motivation for cheating or otherwise break the security of multiplayer games are of course many and different. This is a list of some common motivations, and some examples of how people can, and do, break the security for their own gain. This list is not supposed to be exhaustive, but shows some common scenarios and how the technical means given later in this chapter can be used in practice.

This section refers to the taxonomy in section 3.2.3 and the complete attack graph described in section 3.8 (fig. 3.5).

3.1.1 Win the Game, Defeat Others

People like to win, so performing well in a game is a classical reason to cheat. There are still people cheating in games just for the sake of winning. In addition to feeling superior, cheating can help advance the player in a game where he is not otherwise skilled or patient enough to do so. While cheating improves the players performance, there is also another, less common way to get ahead: destroy the others.

Cheating

Cheating is the most common security problem in games today. When games are single-player only, cheating does not pose a problem as players choosing

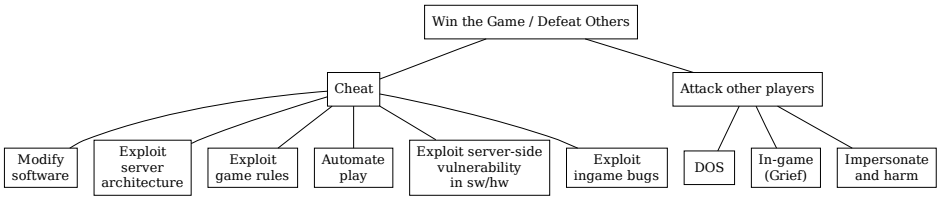

Figure 3.1: Attack-tree for "Win the game, defeat others"

to cheat do not affect others. Cheating in single-player games have a long history, and typically a game would respond to cheat codes that the player could enter to gain in-game items, health etc. Unlike most other types of cheating, cheat codes are created and implemented by the game developer. Another classic type of cheating is to modify saved games. Often single-player games will allow a player to save his or her progression in the game. Typically this will be saved in a separate file, and modifying this file can give the player a different starting point when returning to the game. It is debatable if this is actually cheating. Since the player is not deceiving anyone this can be looked at as playing the game with an alternate set of rules [11, Ch.12].

In multiplayer games cheating is not so innocent, since cheating, either directly or indirectly, affects other players. By cheating a player can gain advantages in-game and become harder or impossible to beat for the other players. Therefore cheating in multiplayer games are not supported by the game developer, and can ruin the game for honest players. Since cheats are not "built into" most MMOs, the cheater has to find other means of cheating.

Cheating is further defined in section 3.2.2.

Attacking Others

Attacking other players is an alternative to cheating for getting ahead in the game. Denial of Service attacks (DOS, section 3.1.3) can be used to disconnect or shut other players out of the game, thereby increasing the attackers chances. Griefing (section 3.1.3) can be used to distract other players and hamper their play. Impersonation can be used as a DOS tactic, for example by closing other peoples accounts. Given the persistent world of most MMOs, it can also be used to ruin other players characters, e.g., giving away their virtual items, or to lower score or rank with abysmal play.

3.1.2 Learning and Exploration

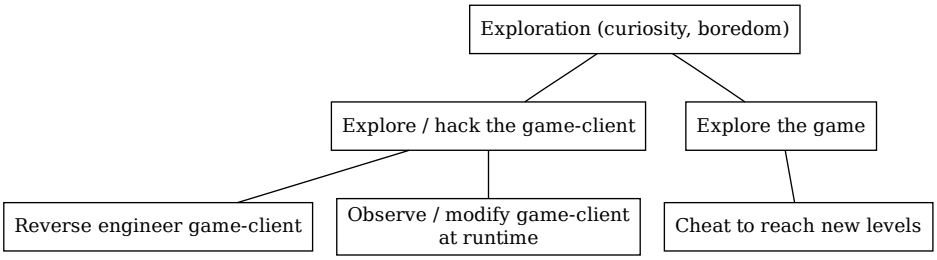

Figure 3.2: Attack-tree for "Learn / Explore"

Curiosity can lead technical savvy gamers to explore and modify their game-client (2b), look for vulnerabilities in the game (4) or the server-side implementation (1b). The motivation can be the challenge it is to break the system or an interest to learn how the game works. Also, players may wish to explore the game world itself, and cheating can be a way to advance in games which would otherwise be too difficult or demand too much effort.

Generally, this type of game hacking should be rather harmless, at least until the gamer realizes the value of his discoveries.

3.1.3 Attack Vendor or Other Players (Malice)

Some people seem inclined to ruin an experience for others, and the (perceived) anonymity of online games often creates opportunities to do so.

Denial Of Service (DOS)

Denial Of Service (DOS) are a non-technical group of attacks where the goal simply is to deny a service, in our example typically the game itself, to others. DOS breaks the availability and can be both a goal in itself or means to an end. As a means to an end such as cheating, DOS can be especially effective when combined with other type of attacks and used at the right moment. An example would be to disconnect a player who is about to win a game or fight.

On the network level Denying service to others can be done in many different ways, and without targeting the network per se. However, flooding

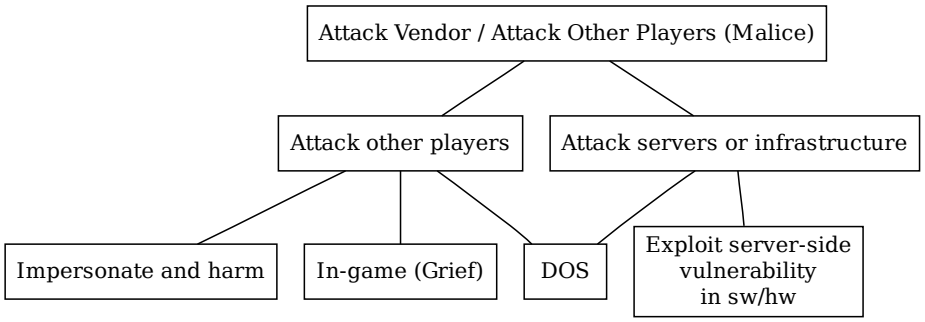

Figure 3.3: Attack-tree for "Attack Vendor or other players"

a host or network with traffic is a classical way to disrupt network applications such as multiplayer games (3b, 3c). By flooding the server or other players' systems it is possible to slow them down or disconnect them entirely. The traffic can be engineered to blend in with other traffic so it is hard to detect, to use specially crafted packets to consume resources at the receivers (network level such as TCP SYN flood or application level) and come from many senders at once (DDOS).

DOS on the network level can also be done by delaying or suppressing traffic (3a iv).

Other types Generally, some DOS attacks misuse security functions in place to lock out attackers. A non-network DOS attack is account-lockout. By abusing the procedures in place to stop brute force guessing of passwords, an attacker can lock out legitimate players by constantly trying to log-in with their username and an incorrect password.

Griefing

Griefing, as in causing grief, is the intentional harassment of other players in multiplayer games. The goal is not to deny service but to degrade the experience for others. The term is used for in-game actions such as attacking one's own teammates or hinder their movement, accusing others of wrongdoing or "stealing" items and experience gained by other players. Abusing the chat function (spamming) can also be griefing.

Griefing is a non-technical term, thus it does not fit well into the technical taxonomy.

3.1.4 Earn Money

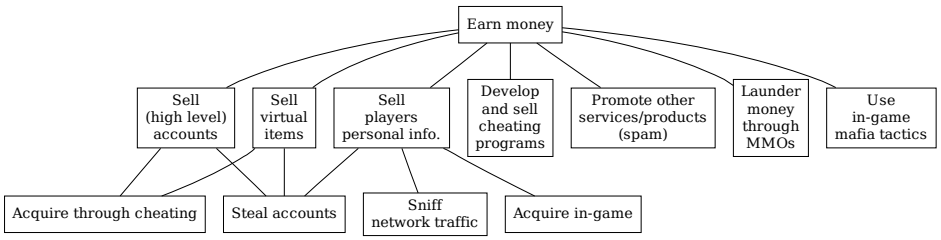

Figure 3.4: Attack-tree for "Earn Money"

In MMOs this is potentially the most important factor, as there are many ways to make money by hacking these games.

The most obvious one might be to cheat in games where real money can be made from virtual items. The cheater gains virtual items faster than other players and these items can then be sold to players who would rather spend some money than the required time and effort to advance in a game.

Personal Information

Personal information is valuable, especially credit card information. If an attacker can steal account information or otherwise hijack accounts (section 3.1.4, account stealing), the players information might be available. In some games personal information can even travel the network unencrypted and is therefore vulnerable to network sniffing (3ai). Encryption does not necessarily help if it is implemented naively, for example, with a static key shared by all game-clients. As with account stealing, this information can also be gathered by exploiting vulnerabilities on the server or in the game-client (1a, 2a), with malware on the client-side (uses 2b), or with non-technical means such as social engineering and phishing.

Games as Communication Channel

Games with a large amount of players and well developed means of communication can find themselves a target for in-game advertisement, i.e., spam. In World of Warcraft for example, spam is a frequent problem in areas, such as the cities, where many people gather. Using the in-game message system is one way, but other, more creative "solutions" also exist. At the time of writing, a popular method seems to be to suicide many characters in such a

way that the bodies spell URLs on the ground. In-game communication is a big part of many online games, so simply disabling it is often not an option. Spam will probably continue to be a big problem for many games, since they provide a relative easy, cheap and risk-free way of reaching many people.

Virtual Assets

A new dimension of MMOs not present in most other games is the connection between virtual value and real life value. Most games only last for a couple of hours or less, but most MMOs are set in a persistent virtual world. This means that achievements stay with a players' character and the character itself improves over time (section 2.2). There is currently money to be made in popular MMOs by selling virtual items, property and characters (section 2.2). This, of course, is a major incentive to cheat or hack these games, since simply playing them normally does not give a good return on the time invested.

Cheat in the Game

By cheating a player can advance in the game, and hence gather virtual assets, quicker. By automating the playing of the game (2c, 2d) a cheater can gather virtual assets without playing. But fully automated gaming, that is a computer program that can play the game without intervention, is difficult to achieve in MMOs because of the complexity of the gameplay. Even so, offloading some of the "work" to the computer can help the player be more efficient, for example by playing many characters at once. Also, games are sometimes played for money, for example a tournament with a cash prize. See also section 3.1.1.

Exploit Flaws in the Game or on the Server

Exploiting bugs and vulnerabilities in the game can also help to gather assets, and sometimes serious flaws in the game architecture or implementation can make it possible to duplicate virtual items. Reportedly, a man in Canada was able to make U.S. $700,000 by duplicating items in Star Wars Galaxies, a Star Wars themed MMO [12, Ch.1].

Steal Virtual Items, Characters etc.

While exploiting flaws in the game, or otherwise cheat, can help to gather assets quicker, simply stealing such assets can be even more efficient. In

MMOs, virtual items are typically bound to a character, and the character is bound to an account.

Account stealing Often the only authentication is a username / password combination, which is validated on the server-side. One way to steal valid accounts is to guess (brute force or manually) this combination (1d, 1e). This information can also be stolen in many ways, such as phishing, malware (2a, 2b), network sniffing (3ai) and fake servers (3d).

Account stealing has become a major problem in WoW. So much so that the developer, Blizzard, are now selling a hardware token ("authenticator") that enables two-factor authentication, but it is optional [13]. If used by the player the username/password combinations is no longer enough to authenticate him, and thus guessing or stealing this becomes less valuable.

According to the security company Eset *"There has been a spectacular increase of malware designed to trick the user into parting with sensitive gaming-related information"* [14]. They also list a trojan designed to steal game-related information as the third most serious threat of 2009 [10]. While the authenticator solves some of the problems, such as attackers sniffing the information on the wire, other problems persist, such as different forms of man-in-the-middle attacks. Recently, a malware surfaced attacking the WoW authenticator by hijacking the log-in process [15].

Develop and Sell Cheat Programs

Developing and selling cheat programs can be another way to make money, as there already seems to be a market for such programs in many popular MMOs. Players who use such cheats may run the risk of being banned form the game, and the developer of the cheat programs might be sued by the game developers.

In 2006 WoW developer Blizzard and the creators of the *Glider* bot software began a court feud in the United Sates. Glider is a program that lets WoW players automate playing of the game (section 3.5.4) and had sold 100,000 copies as of 2008 [16]. Blizzard seems to have won the first round, but an appeal is pending [17][18][1].

[1]One of Blizzards arguments was that Glider facilitates copyright infringement, even though cheating and illegal copying are unrelated. The "illegal copying" supposedly happened when the legal copy of the game was loaded into RAM at runtime. As the Glider user breached the EULA, he or she did not have the "right" to "copy" the game. One would think the fact that this "running is copying" argument held up in court, together with ever-changing EULA's, would have American software consumers running for the hills.

Still, several such cheating programs are readably available[2].

In-game Blackmail and Other Mafia Tactics

There might be an untapped potential for criminal activity in online games as real money stands to be made and the risk is negligible. By hijacking accounts it could be possible to dominate virtual worlds such as MMOs by means similar to those used by organized crime in the real world. In-game blackmail can be possible because there are ways to ruin the experience for a user, take away their character and their income. Although this might not sound so serious, it depends on how involved the user is in the game. Some people will invest a lot of time and money in a MMO, for example have many WoW players been building their in-game characters since the game was released in 2004. The more involved users are, the more difficult it is for them to just walk away, and the more they have to lose by not complying with threats. Other possibilities include pay for protection schemes like the ones operated by the mafia, and the more legit bodyguard counterparts. If there are no enemies, create them. Forcing individuals or groups of players (guilds) to pay a small fee for their "right" to inhabit and move freely in the virtual world seems possible. Similarly, plain extortion is possible as long as the victim has something to lose. Or, one could offer services like guns for hire and hitmen to help defeating enemies and completing in-game missions. There are already reports of services like bodyguarding and possibly even "guns" for hire being offered [19].

Low Risk Low risk is one advantage of committing crime in a virtual world. This has many reasons. For one the virtual identity is not really bound to a person's real identity and thus it is easier to keep them separate and the actual person out of reach of real world law enforcement. In a virtual world it is also easy to operate across borders making it even more difficult to track down and convict perpetrators. Also, a lot of what would be deemed illegal in the real world is not necessarily illegal in the virtual. So far, policing in the virtual worlds are done by the developers themselves and they, of course, have limited powers in the real world. While spreading malware might be illegal, malware is already commonplace and the prisons are not filled with those perpetrators. Actual governmental control in virtual worlds seems non-existent. [20]

---

[2]For example `http://www.mmoninja.com` and `http://www.mmomimic.com`

Launder Money

The lack of control over and regulations in virtual economies can be exploited by real world criminals in an effort to hide the real origin of money and make then untraceable to the authorities. Money is introduced into the virtual world by, e.g., buying characters, gold or other virtual items. The money can be retrieved by selling off these virtual items. By taking advantage of many, possibly hijacked, accounts each amount transfered can be small to further hinder detection, a classic trick used in money laundering. A procedure for automating this would be necessary to do this on a large scale. Money is made by providing this service to real life criminals and others who need to conceal the origin or destination of assets. [20] [21]

3.2 Classifying Problems

Security problems in MMOs is a broad topic, and it is not self-evident how they should be classified. An overview and classification is useful for seeing the "big picture", and as a reference when discussing these security problems and solutions.

3.2.1 Existing Taxonomies

There have been some related work in classifying security issues in games, but most are limited to cheating and does not include other problems. The amount of different taxonomies can be seen as an indication to the problem of classifying different cheats into neatly organized categories.

In [22] and [23] Yan and Randell creates a three-dimensional taxonomy of cheating in online games, and classify cheating into 15 categories. These categories are not disjoint, do not include other security problems affecting games and are not technical enough to fit the purpose of this thesis. They find that the traditional security aspects: confidentiality, integrity, availability and authenticity are insufficient for categorizing cheating, and therefore add fairness as a fifth to contain basically everything that does not fit into the others. It is difficult to categorize cheating based on these aspects. Most vulnerabilities can lead to consequences that fall into many or all of these categories, depending on the viewpoint. Also, one can argue that "fairness" is always breached, since cheating, by definition, is unfair.

Webb and Soh build on previous work by GauthierDickey et al. ([24]) and classify cheats into game, application, protocol and infrastructure categories. They purposely leave out classes of problems they feel are general security problems and not directly game related. Also, their focus is on the difference

between peer-to-peer and client/server architectures, and their infrastructure category lumps together everything outside of the game itself. [25]

Yee et al. [26] focus on MMOs in Asia and do a threat analysis by identifying threats and creating an attack-tree. They identify the following five threats:

1. Gain illegal access to play the game
2. Cheat at game play
3. Disrupt game play
4. Cheat at paying for game play
5. Steal proprietary parts of the software

1 and 4 are basically the same thing and covered as one item in the following list. 2 and 3 are especially interesting due to their highly technical nature. 5 is somewhat interesting, but not game specific. They also create a weighted attack-tree, but it is unclear what they base these values on.

Other taxonomies are the one created by Matthew Pritchard [27] and the one created by Yan and Choi [28] as used in *Cheating in Online Games - Threats and Solutions* by Mørch [29].

The one used later in this chapter concentrates on technical problems, i.e., it does not include social problems such as insider attacks, social engineering, virtual trade fraud and gold purchase. It is inspired by Hoglund and McGraw [12, Ch.6]. Also, I presume that the server is fair and trusted by all parties.

3.2.2 Cheating vs. Non-cheating

One way to classify the different security problems is to divide them into cheating and non-cheating, and thus cheating becomes a subset of the security problems in games.

There is no formal way of defining cheating and several papers come up with their own definition, many of which become quite complicated.

Yan and Randell define cheating as

> Any behavior that a player uses to gain an advantage over his peer players or achieve a target in an online game is cheating if, according to the game rules or at the discretion of the game operator (that is, the game service provider, who is not necessarily the developer of the game), the advantage or the target is one that the player is not supposed to have achieved. [22]

A simpler solution is to simply define cheating as unfair advantage, such as this definition from Webb and Soh:

> We define cheating as a user action that gives an advantage over his/her opponents that is considered unfair by the game developer [25]

Of course this begs the question: what is unfair? There is no general answer that will cover all games. Behaviour that is considered fair and allowed in one game and community can be regarded as cheating in another. The definition of fair can even be different between players of the same game.

Here I'll use yet another definition, and claim that the difference between cheating and other security related problems is not a technical one. The techniques used can be the same, but the expected outcome determines if something should be regarded as cheating. Therefore I define cheating as *"Actions taken to get an unfair in-game advantage"* . By this definition, it would be cheating to exploit a bug in the server software to benefit within the game, but not cheating to exploit the same bug to steal personal information or blackmail the company.

Since this is a non-technical way of classifying the problems it is less interesting for this thesis, since the technical methods and shortcomings are the real issue here, and the intended outcome is less relevant. Often the same technical aspects can be exploited both for cheating, and otherwise break the security, in online games.

This definitions might not be perfect since it portrays even social engineering as cheating if the goal is an in-game advantage.

3.2.3 Classification Based on Target

Another way to classify problems is to divide them into groups based on where the problem technically lies, or, in other words, what is being attacked.

This is a list of low-level technical means, that can be used as a foundation for describing cheating and other security problems in online games. Some items can fit many places and are placed somewhat arbitrary.

1. Server-side
    (a) Exploit software bugs and vulnerabilities
        i. In common software such as un-patched operating systems and services
        ii. In custom software such as the game-server software itself or software handling log-in

(b) Attack game-specific architecture flaws such as failure in synchronisation between different parts of the system

(c) Exploit Hardware vulnerabilities

(d) Guess usernames and passwords

(e) Abuse "forgotten password" functions

2. Client-side

   (a) Exploit software bugs and vulnerabilities

      i. In common software the game relies on

      ii. In custom software such as the game-client itself or included libraries

   (b) Software modification

      i. Before execution

         A. Reverse engineer game-client software

         B. Modify software, such as the game-client executable, operating system or drivers, on disk

      ii. During execution

         A. Read or alter memory

         B. Inject DLL's

         C. Hook Windows userland functions

         D. Hook Windows kernel functions

   (c) Mimicking userinput in software (botting, use on of the above methods)

   (d) Mimicking userinput using hardware

   (e) Hardware attacks

      i. Read memory

         A. Using DMA

         B. Other hardware modifications

      ii. Sniff bus

3. Network traffic

   (a) Proxy / Man in the middle

      i. Observe traffic

      ii. Modify traffic

iii. Generate, insert or replay traffic
iv. Delay or suppress traffic

(b) Flood or otherwise disrupt the players traffic

(c) Flood or otherwise disrupt server traffic

(d) Fake decoy server

4. Rules of the game (the game itself)

(a) Collusion

(b) Score hacking

(c) Game bugs and design flaws

3.3 "Games are software too"

Underlying many of the technical attack vectors is the fact that *"games are software too"* as Hoglund and McGraw note [12]. Therefore the problems seen in other applications often apply to games as well. This includes things like faults in the implementation (software bugs) and faults in the software design (software flaws).

Although these are common problems, some factors can intensify security problems in games. Examples are the complexity of modern MMOs (both architecture and the software itself), pressure to meet release dates, hight tolerance among the users for buggy and incomplete software (games are often patched to weed out problems after the release, and gamers know this). Also, security is not always considered high priority, but I suspect this is quite common.

Many programs in common use have a long history of, more or less, serious security flaws. This goes a long way to prove the often sited theory that no software is perfect. Even seemingly simple client/server programs, for example HTTP servers, often have a long history of flaws and this does not look too promising for games, which are much more complex and less transparent than most software.

There are also many problems unique to games, and those are, at least for this thesis, the most interesting. This thesis focuses less on general security problems in software and more on specific security problems that are relevant for games.

3.4 Server-side Problems

As mentioned, MMOs are often client/server applications. The servers and supporting architecture are running on a trusted system controlled by the game operator or a business partner.

3.4.1 Software Attacks (1a, 1b)

The game servers will typically run both custom game software and common proprietary and open source software. As discussed in section 3.3, complex software is seldom bug free. This class of problems is quite broad and include flaws in the code, flaws in the design and configuration errors. A security breach in the one or more applications running on the servers could lead to modifications of the virtual world, or even full takeover, by a malicious party. An example would be user-input not sanitized correctly leading to SQL injection attacks. Such attacks could of course ruin the database, which would be bad enough, but they could also be used to change something or someone in the virtual world to give the attacker an advantage in-game. But malicious users also take advantage and try to blackmail the game operators.

3.4.2 Hardware Attacks (1c)

Although possible, attacks on the hardware running the server-side game architecture is probably unlikely as it would take considerable insider knowledge to succeed. Examples of this are flaws in different hardware components such as network equipment (routers, firewalls etc.). There are examples of hardware being vulnerable because of mis-configuration, firmware bugs or broken crypto.

3.4.3 Architecture / System Implementation Attacks (1c)

In big games the serverside architecture and the game itself can be quite complex. When the state of the virtual world have to be shared among many servers, inconsistencies can creep in. This is due to the "lag" in communications between servers and the time it takes to actually process the information from other servers. The inconsistency can sometimes be exploited, and it's not feasible to test every possible actions and conditions in complex games. One very simple example works like this: Player A wants to give an item to player B. If this action is implemented naively as a non-atomic copy + delete, the player could try to interrupt the delete, for example by disconnecting.

If successful, the result would be a duplication of the item. Since virtual items can be sold or otherwise converted into real money in many MMOs, the ability to duplicate virtual items is a virtual money press.

3.5 Client-side Problems

Achieving security on the client-side of games is even more daunting than on the server-side. Typically in MMOs the player will run an advanced game-client on his machine, which will connect to the server when playing.

For the purpose of preventing tampering with the client, the user must be considered as the enemy, which means the game-client runs in a possibly hostile environment. Normally, the user has full control over, and access to, his or her own system, while the game only has the level of access granted to it by the user. The game can require high access, such as administrator rights on Windows systems, but it's hard to tell if it has direct control or, for example, is running within a virtual machine.

In most other scenarios concerning security the resource owner applies security mechanisms to prevent unauthorized access, as it is in the best interest of the user to protect his or her own system. It's then up to the attacker to find a crack in the system or the defence, such as a software flaw, a configuration error or a flaw in the design. Unfortunately this has proven to be difficult at times. A case in point is the still widespread existence of viruses and other malware after fighting them for decades. Still, the owner of the system in question does not want malware. But when we put the client part of our game on the users computer, he is already controlling it. As such, stopping, or even detecting, unwanted actions or modifications to the system of the game-client becomes difficult.

This is a fundamental problem in game-client security, and one that is not shared by too many other scenarios. A quite similar problem in many ways is the problem attempted to be solved with content protection, such as copy protection and Digital Rights Management (DRM). With DRM the publishers of licensed digital content, such as music and e-books, try to enforce specific use of the content. The user might for example be authorized to playback a song but not copy or transfer it to another machine or medium. The similarity is that the goal in all of these cases is to give the user some access to the content, but still keep some control. With game-clients it might, depending on the design, be ideal to limit what the users can do with the game-client in order to secure the game. This form of access control is difficult because the user has to be able to, e.g., play the music, but not copy it to another device. Encryption alone does not solve this problem since

the user has to be able to decrypt the content to be able to use it, and the unencrypted content can then be copied freely.

Regarding the client-side, Hoglund and McGraw [12, ch.6] define four types of attack against MMOs and online games in general:

- Controlling the user interface, or *"going over the game"*
- Modifying the game, either in memory or on disk, or *"getting in the game"*
- Using the layer below e.g. drivers, or *"getting under the game"*
- Modifying or generating network traffic, or *"standing way outside the game"*. This can be seen as a client-side problem or a problem of its own, and is covered in 3.6.

3.5.1 Controlling the User Interface (2b, 2c, 2d)

Players control the game through the user interface, typically consisting of a mouse, a keyboard and a monitor. Mimicking user input can be done with either hardware external to the computer or in software. Using hardware it is possible to, e.g., press keys on the keyboard or send signals normally generated by the input devices directly to the input port. Although this is hard to detect if done with hardware, it is more common to do this in software as it is much easier to implement and control. The operating systems own API calls can be used to supply "userinput" to the game-client.

The goal when controlling the user interface is to offload some tiresome, repetitive human tasks, or even facilitate fully automated play. This is discussed in 3.5.4.

3.5.2 Modifying the Game in Memory or on Disk (2bi, 2biiA, 2biiB)

Even online games that connect to a server have to store a lot of game state locally on the client, since transferring all necessary data in real time is not technologically possible. Whatever is stored locally of game code and data, is under the players control. This code and data can be reverse engineered and modified before or after the game is launched. Since the game-client is such a big piece of the overall game, modifying it can change how the game behaves for the attacker.

By reading from memory when the game is running, it is often possible to "see" things the game-client has to know but is hiding from the player. One

example of changing data is to modify your own location in the 3D world, effectively "teleporting" the game character to anywhere in an instant. Unless the server double-checks the client input that is. Assuming it is not possible to move across the 3D world in an instant [3] "teleporting" the character breaks the rules of the gameworld by moving faster than possible. If the server does even the most basic sanity checking on the input it receives from the client, it will detect that this input can not be correct.

3.5.3 Using the Layer Below (2biB, 2biiC, 2biiD)

Like most software, games clients relies on utilizing functionality from the operation system. This makes it possible to fool the game without modifying the game itself, e.g. by letting it load a custom driver. One example of this is the infamous wallhack in first person shooter games where the display driver is modified to make the walls in the 3D world transparent, revealing other players that are suppose to be hidden. [4]

3.5.4 Automated Play / Botting (2c, 2d, 3aiii)

One goal of some of these client-side "attacks" is to be able to automate playing of the game, also called botting. Being able to play more or less automatically has several advantages. Some times the computer can do a better job at playing the game than any human, giving the user a huge advantage if some or all the playing is "offloaded" to the computer. The most common example of this is probably auto-aiming in shooting games, where the player does not aim for himself like he is suppose to, but lets some software do the job for him. This is clearly cheating, as the player gains an unfair advantage, and aiming is part of the game. Computers excel at many different "skills", and whenever one of those are involved in the gameplay, there is a possibility for cheating by letting the computer (software) play parts of the game.

Many MMOs, especially role playing games, have persistent and evolving characters. By playing the game and e.g. do missions or kill enemies the player (or the players character) is rewarded with in-game experience, currency or items. This currency, like in-game money, gold or items have an actual real life value. Therefore it is possible to play a game and later sell the virtual assets. The only problem is that by playing normally one would not earn very much per hour this way, even when exploiting the fact that some

[3] It could be, e.g. with a spell of some sort

[4] There are other ways to achieve similar results, see [30] for an overview

countries have lover average income than others. By more or less automate the playing, one person can play many more characters in the game at once, effectively multiplying his income. The "holy grail" is of course to totally automate the playing, effectively earning money without lifting a finger. This type of botting is also regarded as cheating, and steps may be taken by the game developer to detect and deter automated play. Even so, it still remains a problem.

Technically, botting can be achieved by many means

- Modifying the game itself to automate some operations (2bi, 2biiB).
- Mimic user input (2c, 2d), possibly with feedback from the internal game state or information displayed on the screen. Internal state such as variables can be gathered by modifying the game (2biB), read id from memory at runtime (2biiA) or inject DLLs into the game to run custom code (2biiB). Information displayed on the screen can be such things as the players health or items and spells etc. For most automated playing to be effective feedback is needed. A simple example is a fighter bot that will try to automatically kill non-player characters in a MMO to gain experience, items and money (gold) for the player. Such a bot would, for example, need to know when to attack and when to run away, so monitoring the players (its own) health is crucial.
- Capturing and replaying network traffic to repeat one or more in-game actions, see 3.6.

3.5.5 Combating Client-Side Issues

Solutions currently applied by the game-developers to combat client-side security issues mainly fall into two categories: software obfuscation on the client-side and server-side checking of players action and input from the client. These solutions can be, and often are, combined.

Software Hashing and Signature Based Scanning

One approach to try to stop players from cheating is to use signatures to look for known cheating software on the client. The idea is that if cheats become widespread, and thus a problem, the client can be updated with a signature to match the cheat-code. Valve Anti-Cheat included in games from Valve Corporation uses this approach. Valve games can also hash parts of (or all off?) the game-client, including data files such as sounds, to detect modifications [31].

World Of Warcraft includes a disputed program called the Warden which looks for suspicious programs running on the system. It will compare the window title text and some of the code found in programs running with known values from programs deemed illegal by Blizzard, and also send data back to the servers [12].

The main problem with this approach, other than the invasion of privacy, is that the controlling mechanism runs on the same system as the game and therefore is just as prone to being subverted or modified as the game itself. The server can try to check if the security program is running, but not really know if the answers it is getting are from the actual anti-cheat program, or a fake one. And if it was possible to secure programs running on the client, the anti-cheat program would not be necessary in the first place.

Software Obfuscation

In order to stop reverse-engineering and the modification of software various software obfuscation techniques have been developed. These techniques can be used to secure the game-client itself or a smaller, possibly separate, anti-cheat program. Such techniques usually involve runtime decryption and de-obfuscation, as well as code to try to stop runtime debugging, analysis and modification. Such techniques can be successful in raising the bar for software modification and are often used in DRM and to hamper reverse-engineering of proprietary applications. The downside is that the extra complexity can add new bugs and make debugging the applications difficult. Also it quickly becomes an arms race, and the computer has to be able to execute the program at some point.

There has been some interesting research regarding obfuscation of games such as [32] and [33]. The methods they describe makes it difficult to modify the running software by getting updates from the server and modifying itself. Unfortunately, none of these methods have seemingly been implemented in a complex application, and it would be difficult to do so.

Serverside Checking

In most client/server setups all communication travels through the server and the server is authoritative. Preferably the server should never trust the client, and thus place all security sensitive code on the server and control and sanitize all input from clients. In practice this is often difficult to do, and security alone does not dictate the development.

The server should check the input from the client to see if it is legal, that is, possible according to the rules of the game. In the large virtual worlds of

many MMOs moving from on side of the world to the other can be a chore, hence the popularity of "teleport" cheats that enable the player to move anywhere in an instant. See 3.5.2. Since the server is responsible for informing the client about the (virtual) world around him, such as where other players and non-player characters are, it knows where the player is. It should therefore be simple to detect "teleport" cheats at the server when the game-client suddenly sends a new position far away from the old one. Whenever the game-client flamboyantly breaks the rules of the game - teleporting is just one example - it should be detectable on the server.

Not all cheating is so obvious. If the game-client claims the in-game character is moving at ten times the maximum in-game travel speed, it is clearly lying. But to determine the exact characteristics of an in-game character at any given time takes a lot more work. In MMOs, and other types of games as well, these characteristics are often variable and change as the game progress.

The server, in theory, does have all the information it needs to "rewind" the state of the gameworld to a given time and rerun the operation done at the client. If the results differ from what the client reported, something is amiss.

The main problem with this solution is that advanced checking is very resource intensive. So even if designing elaborate checks on the server is possible, running these checks for each client might not be. Constantly simulating all the clients on the server is not feasible. Also, it will not detect all cheating or fix all security problems such as more passive problems such as information disclosure.

Two possible solutions to this arise. First, there's no need to check all the clients results all the time. Random checks should be enough to eventually catch cheaters. If one random check returns inconclusive results the server can start paying more attention to that particular client. Secondly, it might be possible to offload some or most of this work to other clients. Assuming the other clients are unmodified they will do the servers bidding. Even if some clients are modified and collude with the cheater (or just skips the check to save some CPU time), cheating will be caught eventually, as long as the majority of the players are honest.

3.6 Network Traffic

Network traffic attacks have primarily been used by players against the communication between the server and their own client. But, if a player is able to gain access to other players network traffic, these kinds of attacks could be used against other players as well. The same type of modifications that

can be used to help one player, can be used to hurt another.

3.6.1 Generating or Modifying Traffic (3a)

Normally the server only sees what it receives from the client and has no way of telling what produced the traffic. The game-client will presumably send valid traffic the server understands, but valid traffic does not have to come from the game-client. This traffic could be constructed to be valid or, as in classic replay attacks, be a copy of "old" traffic sent previously. One simple example is to find out what the client sends to the server when the player does some tedious task, and then write a small program that send this traffic again and again. This can then be expanded to generate from scratch more and more of the original game-client's commands, eventually making it an implementation of the game-client.

One real-life example of this is an aimbot created by some students at Stanford for the game Quake. StoogeBot, as they named it, behaves like a proxy and sits between the client and the server. By observing and altering traffic it's capable of aiming and shooting the in-game weapons, relieving the player from such tasks. Since it operates as a separate program there is no modification of the original gameclient. [34] The client simply connects to StoogeBot and StoogeBot connects to the server. Even though StoogeBot is an old program the concept remains valid since there still is no way for the server to distinguish between the original game-client and a proxy without some sort of authentication.

Modifications to the traffic could also be small, such as changing an integer that tells the server how much damage the player did in a fight.

3.6.2 Observing Traffic (3ai)

Instead of changing or inserting data in the traffic sent to the server (integrity), a cheater can just observe the traffic and read out information as is passes from the server to the client (confidentiality). Although observing traffic, in one way or another, is often a prerequisite to modifying it, just simply observing can be easier to achieve.

Just because the server sends this information to the client does not mean the player is suppose to see it, and extracting this information from the network traffic can be an alternative to reading it out of memory or otherwise hacking the game-client software.

Another possibility is to observe other players network traffic to gain information and thus an unfair advantage in-game.

Account information and other information not directly related to the gameworld could leak this way.

3.6.3 Denial of Service (3aiv, 3b, 3c)

See section 3.1.3.

3.6.4 Combating Network Attacks

Most of these problems, except DOS attacks, can be countered by encrypting and signing the network traffic. Unfortunately, encryption adds to the complexity and uses some additional resources. Also the key have to be stored in the game-client, making it retrievable. Having a unique key on each game-client helps as it is then no longer possible read the traffic of other's client. It is also possible to try to hide the key on the client using software obfuscation (section 3.5.5).

3.7 Abusing the Game Logic or Rules

Sometimes it can be possible to exploit the game itself, e.g. the rules or logic of the game. Computer games, like other games, have rules, and breaking those rules can be beneficial to the cheater. The type of attacks that fall into this category are game-specific and it's therefore difficult to give a complete list of these problems. Some of these problems are non-technical in nature and can be design flaws in the game, or problems

3.7.1 Collusion (4a)

Many competitive games, computerized or not, are "vulnerable" to collusion between the players. It can be as a form of information exposure where some players reveal secret information to each other, or other types of teamwork between players who ought to be competing. These players gains an advantage over the honest players.

One type of information exposure can be seen in many first person shooting games where players, when killed, have to wait to re-spawn, i.e., get back into the game. In some games waiting players are "spectators" and can watch everything that is happening in the game, including, for example, where other players are hiding. A player that is killed in-game can share this information with another, active, player to give him or her an unfair advantage. Being a virtual world with virtual identities, one person can join

a game with multiple virtual identities at once[5], without the help of another player.

MMOs are less vulnerable to collusion than many other type of games because teamwork is often encouraged. But teamwork outside the scope intended by the developers can pose a problem. Thottbot[6] gathers information from a large amount of players of World of Warcraft and a few other games. Players voluntary send in information about the gameworld and the site gathers it all into a large database. The system is free and open to everyone, also those who do not commit data, but it is still, in a sense, collusion. If the information was only available to a group of players, it would more resemble cheating.

Collusion is game specific and very hard to combat. The game itself has to be designed in a way that does not reward unwanted cooperation too much.

3.7.2 Score Hacking (4b)

Many games have scoring of some type where the player is rewarded for some actions and punished for others. Depending on the implementation and game rules, it can be possible to fool the scoring system. A common way is to quit or disconnect the game when losing or otherwise being at a disadvantage. For example, if the MMO player is punished for dying in-game, he can simply disconnect if it is inevitable. The server can't tell if the disconnect was on purpose or an error as the gamer can "pull the plug" if necessary.

A real life example, taken from [23], is a weakness in the scoring system of the StarCraft[7] ladder competition. As player was awarded a point for winning matches but not punished for losing matches, two players could collude and play fake matches where they alternately lost to each other. This way both players would win many matches and rise to the top of the ladder. This is example exploits a weakness in the scoring and collusion between players, neither of which are technical problems.

3.7.3 Bugs and Design Flaws (4c)

Design flaws and bugs in games can often be exploited by players who know about them. Although the difference is not always clear-cut, bugs are typi-

[5]For example by running the game on two machines at the same time and joining the same server

[6]http://thottbot.com/about

[7]StarCraft is a hugely popular real-time strategy game released by Blizzard in 1998. See `http://us.blizzard.com/en-us/games/sc/`

cally technical errors in the game such as glitches in the 3D world where walls become transparent. As such, bugs could also be defined as a client-side (2a) or server-side (1a) problems. Design flaws are typically more subtle and are not necessarily technical, such as unintended consequences that arise when players attempt actions not intended or tested by the developers. In role playing games this can arise with spells. Since spells often can be "stacked" or combined, and there are so many of them, bizarre combinations can lead to bizarre results, sometimes giving the player a huge, some would say, unfair advantage.

Due to the complexity of modern games, it is impossible to test every possible combination of actions in-game, and both bugs and design flaws are often discovered by the players after the game is released. Bugs are often quite easy to fix once discovered, but design flaws can mean a major re-design of the game to make it fair.

It is common for players to exploit bugs and design flaws, and such actions are sometimes regarded as a part of the game. It can also be regarded as cheating, even if it's done without extra software or modifications to the game.

[35]

3.8 Attack Graph for Multiplayer Games

Based on common literature and sections 3.2.3 and 3.1, an attack graph for multiplayer games was created. Refer to figure 3.5 on page 39. Motivation (starting points) are marked with gray

It would be useful with a weighted graph to easier see where the main problems lie and what countermeasures would be most effective. Unfortunately there is not enough data[8].

[8]Although I have my own perceptions, the weights are only meaningful if they are somewhat accurate, and preferably based on something more than intuition.

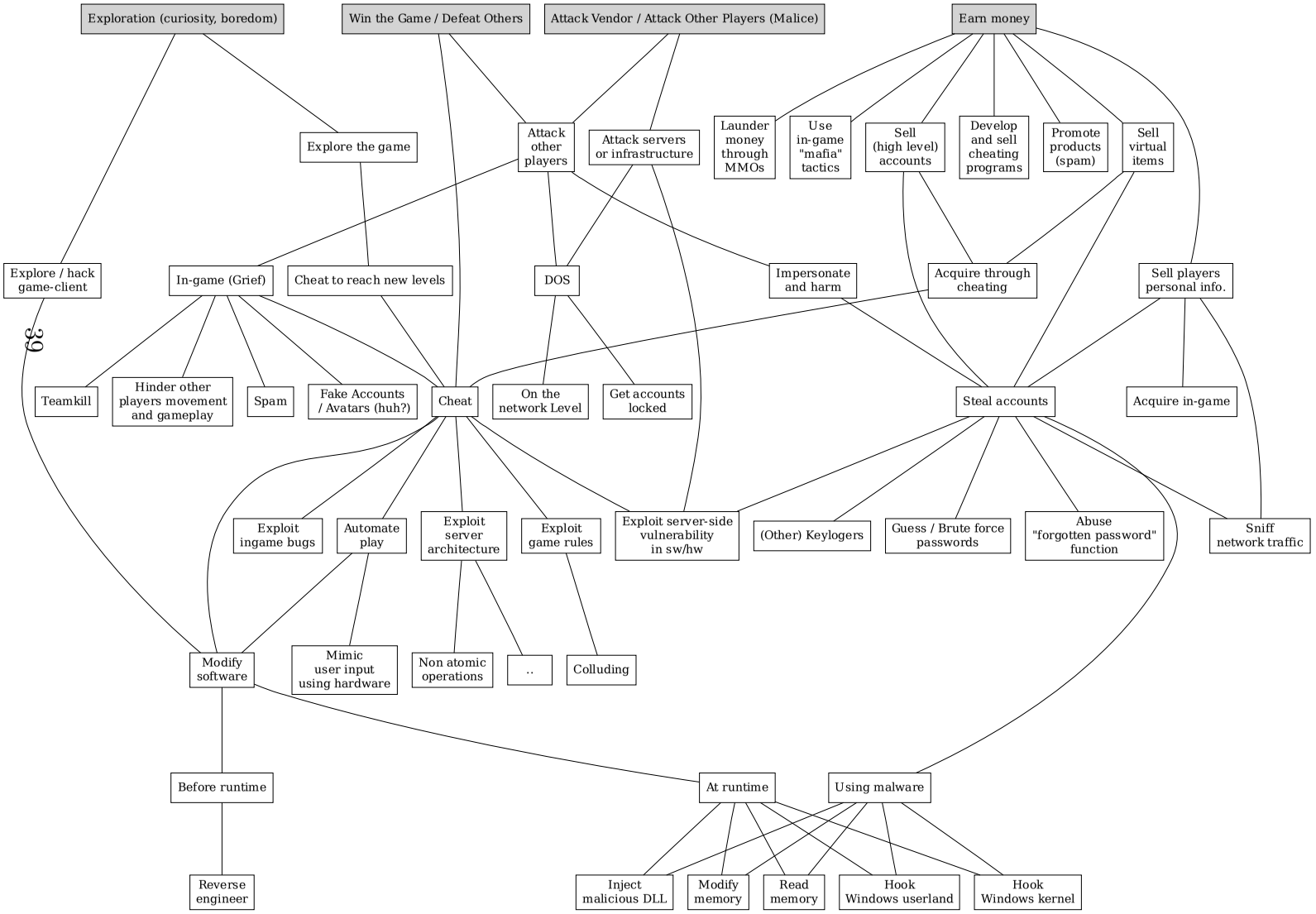

Figure 3.5: Full attack graph for security problems in multiplayer games.

Chapter 4

Potential for Protecting MMOs with Trusted Computing

4.1 Overview of Trusted Computing

Trusted Computing, in the sense it is used here, is a loose term referring to technology developed by the Trusted Computing Group (TCG). The Trusted Computing Group is a consortium that was founded in 2003 by 14 companies including AMD, HP, Intel, Microsoft, Sun, IBM and Sony. It is the successor to the Trusted Computing Platform Alliance, and it's unclear what — if anything — tells them apart from the original Alliance. They now describe themselves as a standards group and have more than 100 members.

Trust defined Trust, as use by the TCG, means that a system or component has the ability to break the security, but not necessarily that it should be trusted, or, in other words, that it is trustworthy. [36, p. 31]

4.1.1 Overview of the Technology

The TPM Specifications

The TCG currently publish several specifications. The first, and arguably the most important specification is the Trusted Platform Module specification, from now on referred to as the TPM specification. The TPM specification is now in version 1.2 and is fairly large and complex, covering the TPM, its use in the platform and software and drivers. The TPM chip is most commonly implemented as a hardware chip attached to the motherboard. Although it is possible (and allowed, see [37, section 4.3]) to implement a TPM in software, it would have to have the same level of tamper protection

as a hardware based solution [37, section 4.3.3]. Since all TPM's in practice will be hardware chips, and software only solutions defy the point, I will refer to TPM as a hardware chip or module.

The TPM Chip

The chip itself is also confusingly called a Trusted Platform Module (TPM). It describes the TPM, a simple hardware cryptoprocessor consisting of a small CPU, memory (volatile and non-volatile) and crypto specific hardware.

The main advantage is the supposed difficulty of modifying hardware compared to software. The TPM is not suppose to be secure against all physical attacks, i.e., not tamper proof. But it is expected to have some anti-tamper measurements to make modifications to the chip difficult. Still, the main goal is to be secure against all sorts of software only tampering or changes. Thus some operations are entirely self contained and not relying of the trustworthiness of other parts or software. These operations should be totally unaffected by software only attacks.

Rationale

As discussed in section 3.5 securing software can be a difficult task, especially when the "attacker" has control over the system, as is the case when it comes to computer games. In general terms, software, when broken, can not be depended upon. To check if software is modified in unwanted ways is difficult when no software can be trusted. An example is rootkits: malware that hides its own presence by modifying the operation system and other software. Detecting rootkits is difficult since the software usually used to check for its existence can not be trusted to return a sincere answer. Having a hardware chip that is untouchable by software creates something that is dependable, even when software is not.

In theory, this neutral chip could be used to let the game server know the truth about the game-client, for example, if the client software had been modified. It could also be used for authenticating the client to the server and vice versa, to secure network traffic and to hide secrets.

The most common threats today — like viruses and other malware — are software only, and indeed has to be for easy propagation. Requiring hardware tampering, however trivial, would be a major gain.

This is also one of the few technologies that are not entirely under the users control, and this property sets Trusted Computing technologies aside from most other solutions for software protection. Although TC technologies do not operate in secret and the user has the ability to enable and disable

features such as the TPM, one of the design goals of, e.g., the TPM specs is to be able to assert the security state of a platform to another. The user can deny this, but not, supposedly, deliver false information. [38, 37, 39, 40] This, in a sense, lack of user control has been criticised (section 4.2), but there are some scenarios where this property is desirable. If malware compromises a machine the lack of a software override is generally a good thing as it keeps the malware from taking total control of the platform. This way it can be detected, at least by other systems. In this case, some sort of hardware override could arguably be implemented without losing security. But made too convenient, and social engineering could be used by malware to fool unsuspecting users to disable their own security. There are at least two scenarios where even a hardware override would break the security: usage controls such as digital rights management and game-client security. Treating the user of his or hers own computer as the "enemy" is understandably controversial, but from our point of view when securing game-clients, the right thing to do.

Activating and Taking Ownership of the TPM

Before the TPM can be used for anything useful it has to be activated, or initialized. This can be done by an administrator, for example in a cooperation where the computers are not owned by the users. Often the users themselves will do this. When initializing the TPM one sets an owner secret which will be used to control the TPM. There is a strong focus on the users control over the TPM so physical access to the machine shall grant the power to reset and re-initialize the TPM, creating a new owner. [38, 39, 40] One way to prove physical presence is to require the user to enter the BIOS setup utility to activate and take ownership of the TPM. Unfortunately the BIOS setup utility is not something most users dabble with, so this will make it somewhat difficult for users to activate and control the TPM.

4.1.2 Promises and Features

The TPM itself is becoming quite widespread REF, but there is still a lack of software truly utilizing the TPM, in the sense that is uses it to do something it could not do in software alone.

"One of the first design goals of TCG was to provide a trusted way to measure and report on a platform's environment [38, Ch.2]

This means it should be possible to remotely determine what software is running on a computer, and if that software was modified.

Another design goal is secure storage for keys and other data. Data stored this way should be protected by the TPM and secure agains all software only

attacks. In a process calles "sealing" data is bound to a given "platform state", and only released by the TPM when the platform is in the given state. This state is defined by PCR values (section 4.1.3).

4.1.3 Basic TPM Functions

The TPM have several components, some are for internal use only but most can be used by outside software.

While the TPM uses a device driver and other software for many operations, some operations are executed within the TPM and thus contained from the outside. This means that software and other devices should not be able to affect the outcome, other than suppling the input.

TPM, both as a specification and as a hardware chip, has existed for some time. Different vendors produce TPM modules and they should all adhere to the specification.

Random Number Generator Random numbers are commonly used in cryptographic operations such as key generation, but are hard to generate in software only because of the deterministic nature of computers [41]. The TPM has a hardware based random number generator that supposedly generate output that is more random than software only solutions. If this is actually the case with todays TPM's remain to be seen[1].

SHA-1 Engine SHA-1 is a cryptographic hash function that produces a 160-bit digest from messages. The TPM is not a cryptographic accelerator and the specification does not specify a minimum throughput for the SHA engine [39]. Therefore SHA-1 engine is primarily for use early in the boot process, before the full operating system is loaded. Higher level software will default to use a software only implementation [38, Ch.7]. It should also be noted that there are known problems with SHA-1. This is discussed in 4.5.

Symmetric Encryption Engine (for internal use) The TPM has an symmetric encryption engine, but it is for internal use by the TPM only, i.e., it does not expose any symmetric cryptographic operations. For data that does not leave the TPM designers use whatever algorithm they wish, but for other use the Verman one-time pad using MGF1 is mandatory. AES can be supported as an alternative.[39, 4.2.2.3]

[1] Or should we say researched. I was unable to find any research proving the effectiveness of the RNG

RSA Engine and Key Generation The TPM is capable of encrypting, decrypting and signing using RSA with keys up to 2048 bits. It can also generate new keypairs and store them internally, i.e., not exposing the private key in the clear outside the TPM.

Pre-Installed Keys and certificates The TPM comes with several pre-installed key pairs. Several certificates are "installed" in the TPM to attest to its implementation. Manufacturers can ship TPM, and thus computers, with more pre-installed certificates than the ones that are strictly necessary. Pre-shared keys can be used to establish encrypted communications over an insecure channel, but authentication depends on the other party's ability to validate the certificate. The certificates are signed by the manufacturer, who thereby vouches for the TPM, that it is genuine and correctly implemented.

Key and Data Storage To keep production costs down the TPM has little storage or memory. Fortunately, the encryption engines and the ability to store keys internally means that the TPM can store large amounts of keys and other data by encrypting it and moving it to the hard disk or other storage device. Although the encryption makes it unreadable (gives confidentiality), it does not stop someone from damaging or simply deleting the encrypted data on disk (availability).

PCR Registers The TPM has several special register called Platform Configuration Regisers (PCRs). By residing inside the TPM, the PCRs have some special capabilities: They can not be read or changed by conventional means (the TPM interface has to be used) and they can only be extended or reset, not set to specific values. When a PCR is extended the old value is concatenated with the new value, hashed with SHA-1, and stored in the PCR. Most of the PCRs are only reset at TPM initialization (typically system power-on) and can only be extended thereafter, and some can be reset by a special CPU instructions at any time.

These PCR values can later be read from the TPM, and possibly signed by the TPM itself using a key stored internally. By checking the signature on the requested PCR values, the other part can be sure the values received are unmodified and was the values stored in the TPM at the time of signing. This process is often called *remote attestation*.

[37]

4.2 Criticism

Trusted Computing and the Trusted Platform Module have received a good deal of criticism and opposition. Some of the most prominent critics of Trusted Computing have been Richard Stallman [42], Ross Anderson [43] and the Electronic Frontier Foundation (EFF)[4]. Another critic of the original TCPA (now the Trusted Computing Group) and Microsoft Palladium (now the Next-Generation Secure Computing Base) was Lucky Green.[44]

They all deal with Trusted Computing using a static root of trust, as explained in 4.3.

Most of this criticism seems to revolve around the users control over his or hers own system.

4.2.1 DRM and Copy Protection

Ross Anderson believes that DRM (section 3.5) was the original motivation for Trusted Computing and Stallman also say that "Hollywood and the record companies" plan to use TC to enforce DRM.

Copy protection is similar. TC can be used to force people to pay for their software.

Although they, or at least Anderson, don't defend pirated software or digital media, they fear that DRM will be used to take away freedoms that customers have today, and that DRM and copy protection is the real reason TC exists.

4.2.2 Vendor Lock-in

Anderson believes that Trusted Computing can be used to "lock" customers to a specific company by. Both Anderson and Stallman uses Microsoft Word as an example: if Word where to encrypt all the files using TC it would be very hard for competing software to read the file, and breaking the encryption might be illegal. He notes that this kind of blatant lock-in might be illegal, but that there are other and more subtle strategies for making it harder to switch to a competing product.

Anderson also believes that TC can destroy free[2] and open source software by making the source code useless. The source will be open, but the user will be unable to use it without an "authorized machine" This, he believes, will destroy the motivation for people to contribute to open source software, and this is one of Microsoft's goals. Stallman notes that TC could be used to

[2]free as in speech

restrict what software a computer could run and that this "puts the existence of free operating systems and free applications at risk".

Although they do not comment on it, there are now some examples of this kind of platform control where the vendor decides what software will run on their platform. Example include Apple products such as the iPhone and the upcoming iPad, and gaming consols such as Playstation 3 and Xbox 360.

The EFF article make yet another point: They argue that the inability of third parties to tell what software you are using is almost always a benefit for computer users. They use Samba, an alternative implementation of Microsoft Windows filesharing[3], as an example. Using Samba computers running other operating systems can connect to Windows servers, or operate as a server for Windows clients. If the Windows machine was able to tell if it was talking to a non-Windows machine it might simply refuse to work or degrade its performance.

4.2.3 Censorship

There also seems to be a fear that TC can be used for censorship by making it possible to delete and retract documents and emails.

In early 2010 Amazon came under criticism for something quite similar to this, although they did it without using TC technology. They remotely deleted some e-books from the Kindle [4] devices of their customers. Ironically at least one of the books in question was written by George Orwell, and some reports claim that 1984 was one of them[5].

4.2.4 Discussion

EFF seems balanced, recognizing that TC have both potential and pitfalls. Ross and Stallman go pretty far in saying that the whole Trusted Computing initiative is a scam to "make your computer obey them instead of you" and stop copying of digital media and software.

The counter argument to some of this criticism is that the TPM is "opt in" that is, turned off by default, and that physical presence will grant the power to reset or turn off the TPM. While this is true, as Green ([44]) note:

[3]Technically an implementation of the SMB and CIFS network protocols. Their website is http://www.samba.org/

[4]Kindle is a hand-held e-book reader sold by Amazon

[5]`http://www.nytimes.com/2009/07/18/technology/companies/18amazon.html`, `http://pogue.blogs.nytimes.com/2009/07/17/some-e-books-are-more-equal-than-others/`

the TPM can turn out to be as voluntary as putting gasoline in your car; no one is forcing you to do it, just don't expect anything to work if you don't.

However, it is not true that the user have to trust the TCG and its members. The specifications are open and public. They do, however, have to trust that the chip is made according to the specification and does not implement extra evil features. The EFF article comments on this. The TPM specifications require the implementation to be certified.

Most users already trust the hardware manufacturers as they have no way to check their computer parts, e.g., BIOS and firmware, but still use it.

The argument that Trusted Computing will help create unbeatable DRM and copy protection might be accurate, but miss the target. The consumers solution to intrusive DRM and copy protection is not to continually break it, but to stop paying for it.

The EFF purpose a change to Trusted Computing where the user can override remote attestation and lie to other systems if they are physically present. They claim this will fix what they call problems with TC. While this might be true, requiring users to use "approved" software — which they deem a bad thing — is exactly what we want when it comes to securing MMOs and other multiplayer games. The EFF recognize this, and even note that their change will render TC useless in securing networked games.

In the end, we have to consider the benefits and potential problems of trusted computing. To do this effectively greater understanding of the technology is needed.

4.3 Trusted Boot, Secure Boot and Static Root of Trust

One of the initial goals of TC was to make it possible to measure and report on a computers environment. This would help better control what software the computer is running in order to prevent unwanted software and unwanted modifications to the platform. This can be done either by stopping unwanted software from running, or just recording the fact that it is so local and remote software can make "meaningful" decisions.

As with many terms used in Trusted Computing, these features and ideas have many different names in different sources, and the terms can be overlapping and even contradictory. Trusted Boot and Secure Boot can refer to just the pre-OS boot process or the whole system with OS and applications. Either way, the goal is to secure the boot process in order to have a secure base to "build" the OS and applications on. Securing the boot alone is not

enough. The OS must implement its own security features to keep control over the platform and stop unwanted software from undermining it, but it can use the TPM and other TC technologies to do so. This is the classical way of building a secure and thrustworthy execution environment using TC, and is sometimes referred to as static root of trust.

Balfe and Mohammed discuss how Trusted Computing can be used to secure multiplayer games in *Final Fantasy — Securing On-Line Gaming with Trusted Computing* [3]. They do not, however describe in detail how this technology works, or how it could be implemented. They also limit themselves to secure or trusted boot, which is an a priori requirement for all their purposed solutions.

Trusted Boot vs Secure Boot

The difference between "trusted boot" and "secure boot" is, according to [38, Ch.2], that secure boot will not allow the system to boot into an untrusted state, presumably by stopping the boot process whenever it finds something amiss. Trusted boot on the other hand just records what was booted so that it can be reported later. This does not stop unwanted or modified software in the boot process, but reports the fact that it's present. Trusted boot seems more pragmatic since it never stops the user from booting his system, but allows him to prove that he booted in a certain way. Trusted boot is sometimes called authenticated boot.

Chain of Trust

The TCG uses the transitive relation on trust, meaning that if part A trusts part B, and part B trusts part C, then part A trusts part C. Trusted and Secure boot uses this to build a "Chain of Trust" by starting with a trusted starting point and extending the trust to other parts deemed trustworthy.

Benefits

By controlling the system from the startup and never letting go of this control it's possible to control what software is running on the system.

In recent years there have been several proof-of-concept rootkits developed for PC firmware, such as the BIOS and expansion cards. This has not been a huge threat in the wild, probably because there are lower hanging fruit, but the concept is valid. Such rootkits escape the most common detection techniques and can survive operation system upgrades and re-installs.

It can be hard to implement a rootkit in firmware since it runs so early in the boot process and none of the high level api's are available, and there is no other programs running to interfere with. But a rootkit could modify content on disk, such as the OS or other software, and set it self up in memory. Also, some firmware is executed after the OS has loaded, for example ACPI code in the BIOS and some legacy features in the video card. John Heasman have shown examples of both these techniques[45]. Other possibilities are to modify the MBR, VBR, boot sector or bootloader on the disk drive [45]. Correctly working secure boot or trusted boot would stop or detect these attacks.

4.3.1 How it Works

Trusted Computing covers many different hardware platforms. Implementation of trusted boot is platform specific and what will be described here is for personal computers[6].

The success of Trusted Boot and Secure Boot depends on a few abilities:

1. A known secure starting point or state. Typically, this will be the state where the computer is turned off and no software is running.

2. A way to measure or check the next part of the boot cycle. The PC platform boot is a multi-stage procedure, and each stage has to be measured or checked before it is handed control over the CPU and other hardware.

3. A way to store the measured values. For trusted boot, these values have to be stored in such a way that they can not be modified by subsequent stages in the boot process. This is because control can be handed to stages found to be "bad" or modified, as the goal is not necessarily to stop those from executing but to know that they are not trustworthy.

   For secure boot, these values have to be stored on forehand in order for the check to happen at this time and not later.

4. For trusted boot, a way to read those values out, either for local or remote software. Integrity of this data are the main concern not availability. This means that "bad" parts or code might be able to stop those values from being read, but it should not be able to fake or change them in any way.

[6]IBM PC compatible hardware

5. Some knowledge of "good" or correct values. Reading and reporting these values does not accomplish anything unless we know what they are "suppose" to be.

We will now look at how these prerequisites can be fulfilled with trusted computing.

Secure starting state The idea is that the computer is in a known and secure state before it is powered on. Unless the platform has been modified, it is known what will happen when it is turned on: the CPU in a IBM compatible PC will start executing the BIOS from a hard-wired address. In a TC system, the control is first given to the core root of trust measurement (CRTM). The CRTM should preferably be a part of (located inside) the TPM chip, but the specifications allow it to be physically separate, and todays motherboards have the CRTM as a part of the BIOS. Even if the CRTM is located outside the TPM chip it is trusted and thus it should not be easy to modify the CTRM itself, or to bypass it or the TPM during boot. Thus if the CRTM is part of the BIOS, this initial BIOS code must be immutable and not reflashable like the rest of the BIOS normally is [38, ch.6]. Although physical modifications are not entirely preventable, the TPM specifications do specify that the manufacturer have to implement tampering countermeasures. How well the CRTM is protected in todays implementations is an open question. According to Kauer[46] at least one computer they tested did not follow the TPM specification and protect the BIOS and CRTM (section 4.5).

Measuring When the CRTM gets control it is its job is to measure the rest of the BIOS and hand control over to it. "Measuring" is most easily implemented as a SHA-1 hash of the binary, but it is possible to devise other ways of measuring code and data, and the measuring in each step does not have to be the same. It is then the job of the BIOS to measure the firmware located on expantion cards and other hardware, and the MBR. Then the MBR measures the bootloader, the bootloader the OS kernel, the kernel the rest of the OS. At this point it is the OS's job to measure any applications before they are executed. It's debatable if this is part of trusted and secure boot, but the boot lays the foundation for the OS.

Storing Measurements For trusted boot, where these measured values are to be used later, the values have to be stored after each step, in a way unchangeable to the subsequent steps. Or, more precisely, it has to be stored in a way that makes change apparent to everyone. This is exactly what the PCR's in the TPM accomplishes, see 4.1.3. The properties of a cryptographic

hash function like SHA-1 (but see 4.5 for a caveat) means that it is very hard to find a value that will hash to a given value, and therefore for a piece of software to "undo" the PCR to a good state, even if the software knows what this good state is. To do this, it would be necessary to reset the PCR and extend it with the good values until the correct PCR value is obtained. Therefore the PCRs used in trusted boot can not be restet after the TPM is initialized. One way of thinking about this is that we are storing the executables themselves, as the hash is one-way and as good as storing the binaries themselves. In addition a log or history of what has been measured and the value obtained for each PCR can be stored outside the TPM, for example on the disk drive. The history file does not have to be secure against tampering because the PCR's and the values in the history file have to match, so altering just the history file accomplishes nothing.

For secure boot, it is not necessary to store the measured values as part of the boot.

Reading and reporting PCR Values The PCR values are stored in the TPM and can be read out with low or high level TPM API's.

For secure boot, this is enough since software can trust itself; was it not "secure", the boot process should not have reached it. Reporting PCR values to another system (third party) is not necessary for secure boot, but might be desirable after boot to prove to other parties that the system was booted securely.

The PCR values are stored in the TPM and can therefore be signed by the TPM itself and sent out. The signature should be generated by a key internal to the TPM. How this is perceived to be done in TC is discussed later in this chapter.

For trusted boot, checking the local client is different because there is no known environment to fetch the PCR values in a secure manner. Even if they are signed by the TPM, there is no obvious way of checking the signature locally; if the system have been attacked we can not trust the software to check the signature. A workaround is to not read of the PCR values at all, but let the TPM handle the comparison.

Locally comparing to known good values When using secure boot the comparison is done at boot time: each part of the boot process is measured and compared before executing. This means that this low level code (real mode for most of it) must implement code to both hash (or otherwise measure) code and talk to the TPM. This could probably be implemented in the CRTM or BIOS and shared with the other code. A problem with

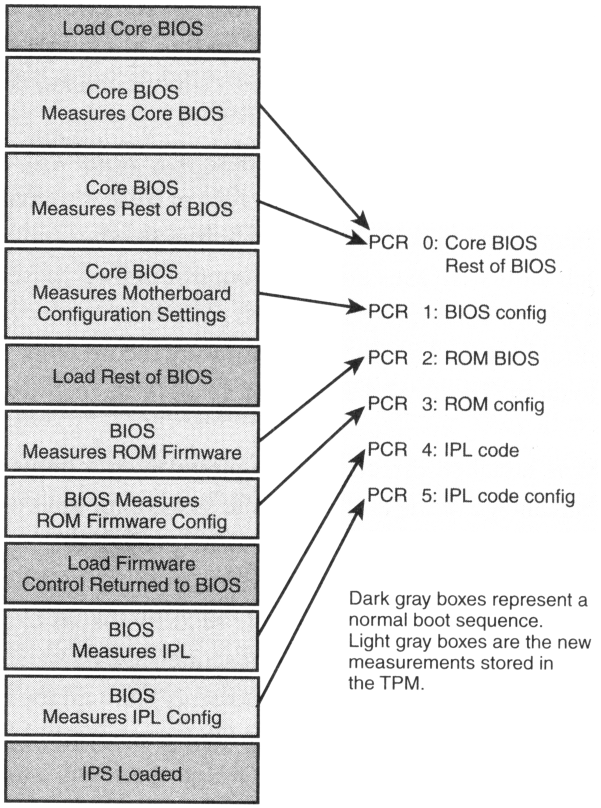

Figure 4.1: Usage of the PCRs in a PC trusted boot. How code executed during the boot process is extended into different PCRs. From [38, ch.2].

this approach is that the same chain of trust that will stop tampering with the boot process will stop upgrading. A solution is to switch to public key cryptography as early as possible, as seen with the Xbox 360 (section 4.3.2).

In a process called sealing, the TPM can "lock" data to one or more PCR's. The data will only be released, or available for use by the TPM, if the PCR(s) have the correct values. The correct PCR values are defined when the sealing is done. For trusted boot, a way to check if the local system has been booted in a trusted way is to seal a secret to a PCR and then the software can try to use/get this secret. This could for example be a key for symmetric encryption. If we are able to get the key and decrypt the data we know the PCR(s) have a previously defined good value [38, ch.2]. Microsoft Bitlocker uses a variation of this: the volume containing the operation system is encrypted with a symmetric key and the key is sealed to "good" PCR values.

One way to learn the "good" values is to take the values recorded at first boot, or another point in time, as correct. Bitocker stores these values when Bitlocker is first enabled.

Remotely comparing to known good values For trusted boot, the history created when booting contains all the steps (extends) that has happened to each PCR. When the PCR values are reported to another system along with the history file, the other system (e.g., a game server) can check that it only contains approved software. If it does, the PCR values can be checked against a database of correct values, or the other system can do the same "extends" steps on the correct software and compare the answer to the PCR values received.

A combination of local checking and remote checking is probably desirable. If the OS could attest that it loaded securely, it could just send over a list of software executed since boot. Or better yet, do this check locally as well, since sending a list of software can be seen as a privacy issue[7].

4.3.2 Examples of Secure Boot and Trusted Boot

Microsoft Bitlocker

The Microsoft technology collectively known as Bitlocker is a system for drive encryption which also, in some modes, builds a chain of trust to help secure the data. Refer to figure 4.2.

Bitlocker uses a combination of trusted and secure boot. Windows will not load if there is a problem during boot, but the boot will not halt at the first sign of trouble. A true trusted boot setup would let the OS load but record the fact that the boot was not secure. Also, only the boot process up and until the boot manager is checked, once the boot manager unseals the symmetric key, Windows loads normally and have the responsibility of keeping the system secure.[47]

A problem with the Bitlocker approach is that it stores values from first boot after its enabled as good, and seals the key in the TPM with those PCR values. An attacker (boot rootkit) can therefore reset the good values and break the system.

On the one hand, an attacker with enough privileges to install a boot rootkit from software could also turn Bitlocker off or steal the data from the running system, so this is not too serious. On the other hand, this ruins Bitlocker for our purpose since a Bitlocker secure boot can not guarantee the integrity of Windows.

[7] Although Blizzard seems to get a away with something similar with the Warden

Also, they seemingly forgot to include expansion cards (such as PCI, PCIe, AGP) in the boot process check. If this is actually the case, or if it's just an oversight in the documentation, is unknown. Heasman[45] explain how code from expansion cards sometimes are loaded into RAM and execute by the BIOS.

Another potential problem, although unrelated to our purpose, is the leakage of the symmetric key from RAM on a running system. MacIver mentioned this problem in 2006 [47][8]. [49]

The Microsoft Xbox

The original Xbox video game console, was produced by Microsoft and released in 2001. Built upon commodity PC hardware with some modifications, and ran a modified Microsoft Windows operating system.

Pirated games is an issue with console (as well as pc) games, and cheating

[8]This has been a known problem for some time, and later the infamous "Cold Boot attack" brought this issue more attention [48]

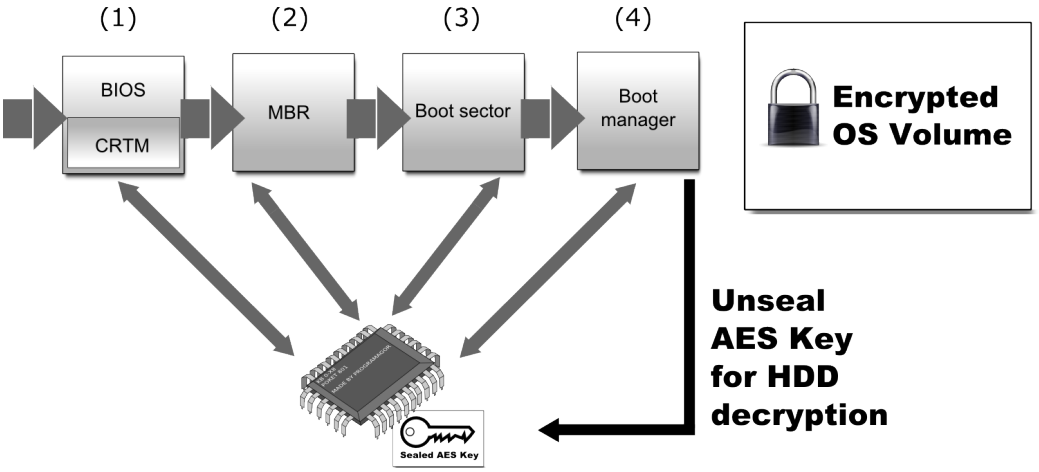

Figure 4.2: A simplified view of Microsoft Bitlocker operation. (1) When the system starts the CPU starts executing the code in the CRTM. The CRTM extends a PCR in the TPM with the MRB. (2), (3) This pattern continues through the different stages of the boot. (4) The boot manager will unseal the symmetric key needed to decrypt the disk content. The TPM will only release the key if the PCRs have the correct value, i.e., the values stored when Bitlocker was first enabled. If successful, the boot manager can decrypt the disk and hand execution over to the boot loader, which will load the Kernel. [47].

and malware is a potential problem on gaming consols as well. Also, Microsoft earns money selling games and reportedly lost money on each Xbox they sold in the beginning. Therefore they would like to avoid that a large number of sold Xbox consols were used as cheap PCs instead of gaming consoles [50]. Xbox features a secure boot like boot in order to "lock" the platform down and disallow any unofficial games or software to run. This booting procedure of the Xbox uses the same "chain of trust" principle as the trusted boot used in trusted computing. Since the Xbox is so similar to "normal" PC hardware, the solution created by Microsoft is especially interesting.

The Xbox does not use a TPM, but build their chain of trust by other means. Refer to figure 4.3 on page 57.

1. The root of trust in the Xbox is a 512-byte secret boot block that is hard coded into the southbridge chip. The Xbox boots from this code but tries to hide this fact with decoy boot code.

   This decoy boot block is located in a flash ROM chip and is presumably included to obscure the booting process and hide the real boot code in the southbridge, and therefore contains a "halfway reasonable looking decryption and initialization code" [51]. Another possibility is that the decoy boot code was not meant to be a decoy at all but was unintentionally left in the flash ROM.

   After some system initialization the secret boot block reads the kernel bootloader from the non-volatile flash ROM chip. The kernel bootloader, encrypted with the RC4 cipher, is decrypted. The key is stored in the boot block. After decryption, the boot block will try to verify the kernel bootloader by looking for a 32-bit number in the decrypted content. [50]

   This check makes it possible, intentionally or not, to update the kernel bootloader.

2. The boot block jumps to the kernel bootloader. The kernel bootloader, now decrypted and verified, performs some more system initialization. The kernel bootloader, using a key stored in the bootloader itself, decrypts and verifies the kernel, stored in the same non-volatile flash ROM. This verification is necessary because the flash ROM, although non-volatile, is re-flashable and should not be trusted.

3. Once the kernel is loaded it is its job to decrypt and load necessary libraries and execute games from DVD discs and other applications stored on the internal disk drive. All games and applications must be

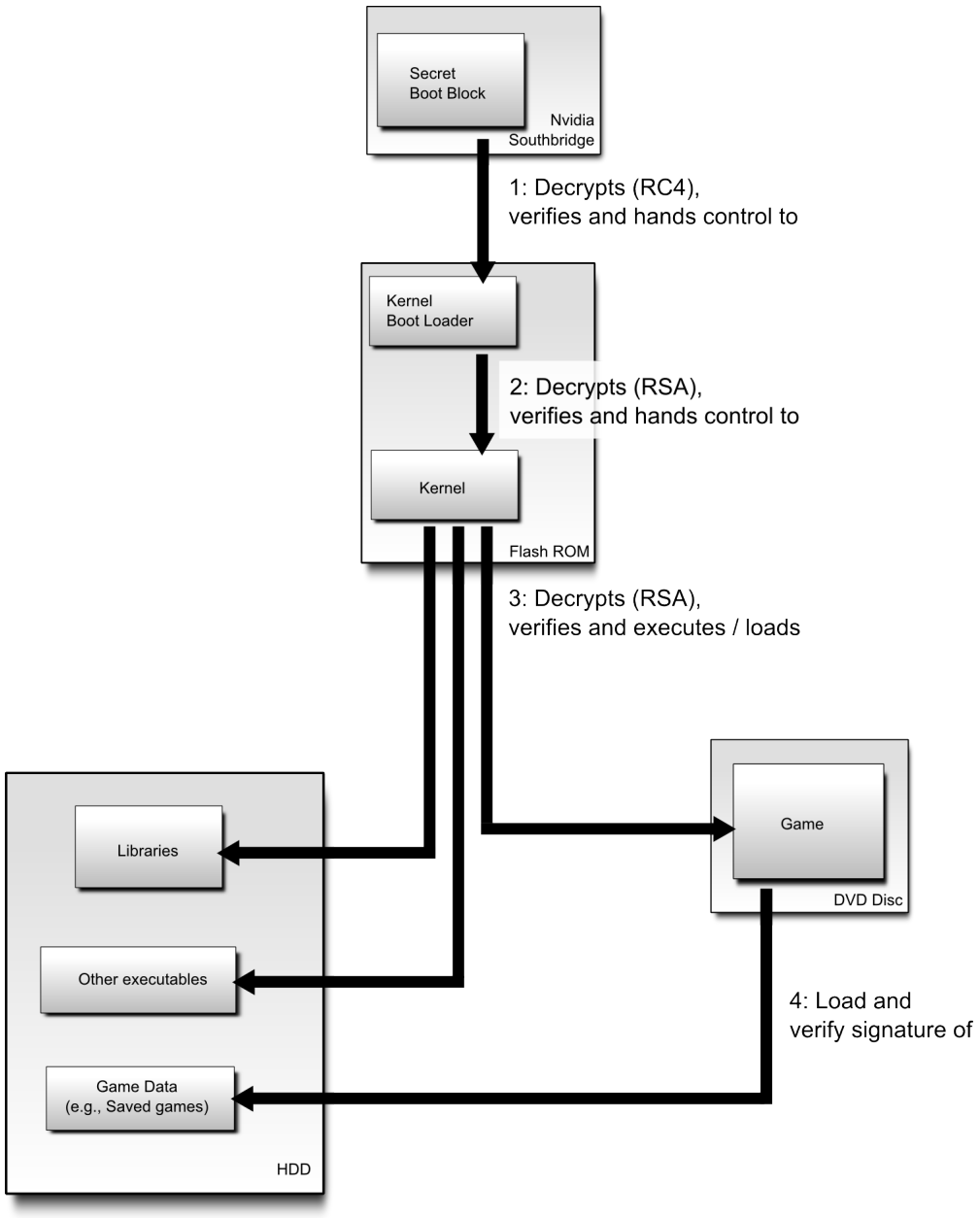

Figure 4.3: Overview of the Xbox security features and chain of trust: (1) The Xbox boots from a secret boot block in the southbridge, which decrypts and jumps to a RC4 encrypted kernel bootloader which (2) decrypts and jumps to the stripped down RSA encrypted Windows 2000 kernel. (3) The kernel executes games and other binaries, and loads necessary libraries, if the are signed with correct RSA keys.

signed. The appropriate keys for signature checking and decrypting are located within the kernel.

4. Games can load custom data, such as saved games from the disk drive or USB storage. Saved games let the player return to a place in the game without having to start from scratch each time.

The security of the system comes from the immutable initial bootcode in the southbridge. It is not easy to replace the chip, overwrite the code in it or override execution of this code.

Even if the flash ROM that stores the kernel is reflashable and thus not secure, it can not be modified without the initial boot block detecting this and refuse to execute it. The chain of trust is further build by using RSA encryption and signatures, which should be checked on all code before it's executed.

Several flaws in the design and implementation of the Xbox "trusted boot" made it vulnerable to attacks.

- Since the secret boot block is contained in the southbridge, it has to communicate with the CPU over a bus. With a boot block of only 512-bytes and commodity parts, any sort of encryption on the bus is difficult. But the bus is high-speed and they presumably assumed it would be very difficult to tap. Unfortunately for Microsoft, Huang built custom hardware and read the bus and thus the content of the secret boot block [51].

  One solution to this problem is to move the boot block and make it a part of the CPU, but this would mean stock CPUs could not have been used. Another solution could be to include a CPU with the code running the en- and decryption, which, to some degree[9], is the case with the TPM.

- The check used by the boot block to verify the kernel bootloader is simple and anyone with knowledge of the key, the "magic" number and the cipher (RC4) can create their own bootloader. Huang theorises that this check is so simple because of the limited storage space in the southbridge [51] while Domke and Steil points out that the RC5 cipher used in the decoy boot block behaves differently[10][50]. It is also

[9]The TPM internal RSA encryption is available for use by software, but the TPM is not a crypto processor and will be slow compared to the main CPU in most cases. Therefore, a developer might opt to do all encryption for his software without the TPM.

[10]They seem to belive that the use of the blockcipher RC5 in place of the streamcipher RC4 would fix this problem, but this is not evident.

possible that this was designed this way to make it possible to update the kernel bootloader.

If one can trick the boot block to load another kernel bootloader the chain of trust breaks and the whole security crumbles. It is the kernel bootloader's job to verify the integrity of the kernel, and if the kernel bootloader can be modified or replaced, the kernel can be modified or replaced. This, in turn, makes it possible to load another operating system or run a modified kernel that, for example, does not authenticate games.

This attack hinges on the ability to read the secret boot block containing the key.

- Over time several arcane features and quirks of the PC platform and hardware used have been utilized to break the chain of trust created in the Xbox. One of them, the Intel "bug" exploits a difference between Intel and AMD CPU's: The Xbox was originally suppose to ship with CPU's from AMD but Intel was chosen as the vendor late in the developing process. The Xbox expects the CPU to throw an exception when reading through end of memory, but the Intel processors does not work this way and continues reading at the start of memory. This makes it possible to get the Xbox to execute custom, unchecked code. [50]

- The DVD drive checks if discs are authentic and is trusted by the kernel. But the firmware is not protected and can be overwritten with a version that confirms all discs [52].

- A software only attack is also possible. The chain of trust is broken in many games because they load saved games without checking/measuring them correctly first, and have a bug in the load routine.

  Also, these saved games can be data created at run-time by the game software, and the game software has to be able to sign saved games. This leads to the private key being kept in the game software [50]. Even if the games are encrypted and the chain of trust is intact, it would be hard to keep this secret key out of RAM, where it could be read barring other security measures.

  A TPM could help mitigate this problem by signing and checking the saved games. To sign using the TPM requires the ability to run custom code on the system, which is prohibited as long as the chain of trust is not broken.

> Games are complex pices of software - even if the chain of trust holds up to this point - expecting them to be bug free is arguably naive. All games run in kernel mode and full access to the system is given to the game, and whoever can successfully exploit one.
>
> [50]

Domke and Steil argue the Xbox was rushed to marked and the security system was devised by people without any real security expertise [50]. This could explain some of the failures made with the Xbox, but not all the failures are obvious, and these can be taken as a hint to the difficulty of creating a chain of trust using commodity hardware.

The Microsoft Xbox 360

The Xbox 360 have several differences from the first Xbox. While the original was built with commodity hardware, the Xbox 360 utilizes components not normally found in personal computers. It runs on a customized IBM 64-bit PowerPC CPU with three cores. Although it does use common technologies such as PCI and SATA, several legacy technologies have been dropped.

To increase the security several extensions have been included on the CPU die. Refer to figure 4.4 on page 60.

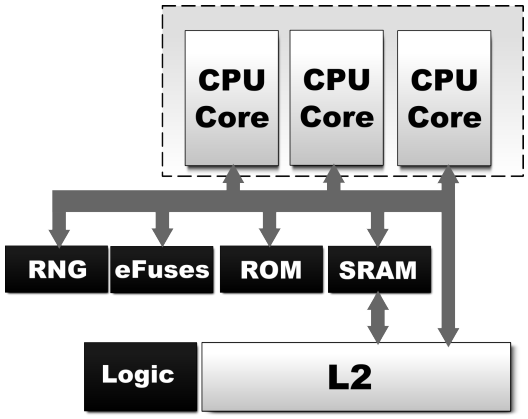

Figure 4.4: A simplified view of the Xbox "Xenon" CPU, based on [52] and [53]. Marked in black are the parts added for the Xbox 360: A random number generator, eFuses, ROM (32KB), SRAM (64KB) and Logic for en- and decryption as well as hashing page tables.

Several changes to the design was made from the first Xbox:

- The ROM containing the boot code and 64 KB of RAM in now included on the CPU die (fig. 4.4). This removes the need to send data from the boot block over a bus, and the ROM is large enough to hold the necessary code to do "real" checking of the next part of the boot process.

  The internal RAM makes it possible for the CPU to store limited amounts of data internally instead of in system RAM. This is used during the first stages of booting, when encrypted ram is not yet usable.

  The bus-taping attack used against the first Xbox will not work simply because the boot code does not travel over a bus anymore.

- EFuses, an IBM technology, have also been added to the CPU. There are 768 eFuses and each one represents one bit: it's either logically (and physically?) "burned" or not. Both "burning" an eFuse and reading its state can be done from software. Restoring an eFuse can not be done, at least not from software.

  The eFuses are used to "burn" a unique (for each Xbox) serial into the CPU which is also used as a key for symmetric encryption.

  They are also used by some software to stop downgrading: When executing the software will check an eFuse to see if it's allowed to run. An update can "burn" this eFuse to stop the old version from running.

- To stop important data from "leaking" from system RAM, all code, and the data deemed sensitive is encrypted at runtime before it leaves the CPU. The key used for this is randomly generated at boot.

  Encryption does not stop the destruction of data. Domke and Steil [54] uses a blacklist stored in RAM as an example. The blacklist would be needed for revocation, but could easily be destroyed by flipping a bit in the encrypted data. Although changing encrypted data will yield unpredictable decrypted data, it can still be harmful. Especially if the encrypted data is code and it's possible to modify small portions. A simple example would be modifying the target of a jump operation. The execution would jump to a "random" address, but with some patience and luck it could be made to jump to an address space controlled by the attacker.

  Important segments, up to 1 MB, can therefore be hashed as well. The 1MB limit is due to the hash values being stored in the small CPU RAM.

The hashing, en- and decryption is implemented in (besides?) the L2 cache one the CPU die.

- Games are now running under a hypervisor to control and limit what they can do. Most of the hardware is accessible for the games directly. The hypervisor reserves the right to do security sensitive operations such as cryptography, but make these operations available to other software with syscalls. This means that the system no longer trust games fully, even if it should not be possible to modify games in any way or run other custom code.

  As mentioned, games are complex and might contain exploitable bugs. The use of the hypervios means that even successfully exploited software can not "roam free" on the system.

  The hypervisor is running with the highest privileges and is, reportedly, small and well written [54].

- W^X (NX) is also implemented in the hypervisor. Each page can be either writable or executable to make it harder to exploit software bugs such as buffer overflows: Code injected can not be executed. Theres no hardware support, the hypervisor has the full responsibility of enforcing these checks, and only the hypervisor can mark a page executable.

Booting:

1. Booting starts with the boot code stored in the CPU ROM. It will decrypt and check the next stage bootloader from a flash ROM chip outside the CPU. This boot code can not be updated since it's stored in the CPU ROM. A flaw here would be disastrous.

2. This loader checks an eFuse to see if it has been revoked, sets up the external (system) RAM, and the en- and decryption using a fresh random key. It will then decrypt, check and load the next stage loader into external RAM (encrypted).

   This is the first code which checks to see if itself has been revoked, through an eFuse. This sounds bad, since it relies on software checking itself, but actually works as long as the check is before the exploit in the program.

   Generally, the reason for revoking code is to stop downgrading. When a bug is found in a piece of software, that software can be fixed and upgraded. In the Xbox 360 all software, except the first boot block, are stored on the flash ROM chip and thus can be upgraded through

flashing this chip. If the Xbox will actually flash this chip itself or if an upgrade only affects new units, is unknown. In this case, downgrading would work by re-flashing the flash ROM with its old (pre-update) content after an upgrade. Since this old content is indeed signed by the correct keys, it would be allowed, if it where not for the eFuse checks. After a downgrade the boot loader will count patches included on the flash ROM, and compare the number to a special batch of eFuses. There will be fewer patches than the number of patch eFuses "blown" and it will halt execution. This assumes the code with the bug is executed after the patch check, if the bug comes before the check it is possible to hijack the program and thwart the check.

The system halts execution on security errors. In trusted computing it would be useful to be able to continue to load the operating system, but flag the fact that it is no longer "secure" and trustworthy, i.e., trusted boot. In the Xbox 360 this is not possible since the CPU can not sign arbitrary date like the TPM can, just memory pages. This is needed to be able to respond to a challenge from a remote system with the status of this flag, signed by the CPU itself. The Xbox 360 also lacks a storage facility like the PCRs to store the "we have been hacked" flag in an unchangeable manner. "Blowing" an eFuse would work, but they are not restored at reboot, so a glitch in the boot could render the system permanently "hacket" to the outside world.

3. This kernel loader checks, decrypts and extracts the kernel into encrypted system RAM. The kernel is never changed or updated, but the kernel loader can apply patches (from flash ROM) if available.

4. As with the first Xbox, the kernel (with supporting libraries) is responsible for checking games and other software before launching them. This is done with RSA keys and signatures.

   Even games and software found to be "good" are not given full control over the platform since the hypervisor runs with higher privileges.

Although a better design the first Xbox, the 360 is not perfect, and a few successful attacks have been launched against it ([54, 52]) :

- There was a bug in the hypervisor, the code that runs with the highest privileges.
- The CPU serial burned into eFuses is used a symmetric key for some operations. By connecting to the correct bus a timing attack on memcompare, which is used to compare the values, is possible. Since the

serial is unique, this is not a class attack and has to be done on each Xbox.

- Games does not sign game data, in itself not a problem, but a special kind of "game data" called "shader code" is actually executed. It should therefore be signed but, is not.

Other questions also arise:

- The Xbox 360 security very much rests on the CPU and its ability to keep secrets. The CPU die is often regarded as untouchable. Is it? Presumably the CPU was not built with anti-tampering in mind. The small size might render disassembly "impossible" but other attack vectors are available. Attacks have been seen on other chips where the working condition of the chip (such as voltage or temperature) is altered to make it behave in ways it was not meant to.
- At least one person claim to be able to disable eFuses by disconnecting pins on the CPU[11].

4.3.3 Challenges

Although sometimes overlooked in TC literature, we face many challenges when creating and maintaining a trusted boot. The challenge discussed here are specific to creating a chain of trust with trusted or secure boot. A more general discussion about TC can be found in section 4.5, and problems directly related to MMOs are discussed in 4.6.1.

Needed software changes Trusted boot and secure boot require more than just a TPM. Depending on the implementation it also requires changes to some of, or all of, the stages in the boot process. This includes the BIOS, MBR, VBR, expansion cards, boot loaders and operation system itself.

Knowing the right answer When a secure boot solution is attempted, the firmware and software included must know the correct values a priori, which hampers the upgrading of individual parts of the system.

Trusted boot, instead of secure boot, allows the user to use the system in whatever way he or she wants and only worry about the security status when using TC enabled software. Even so, for trusted boot to become useful in normal settings[12], it has to be user-friendly and not "complain" too often.

[11] See: http://dwl.xbox-scene.com/tutorial/XBOX360cpu15data.pdf

[12] Settings where security is not a high priority.

The most promising solution is to sign the PCR values and log with a non-migratable TPM key[13] and ship them off to another system for verification. This system would then have to know the correct values for the hardware (firmware really) and software versions used. Since PCs are so diverse, gathering the required information will be tough. Fortunately, gaming systems a are probably a little more congruent than most systems. Also, Microsoft already gathers vast amounts of information about drivers in their driver signing program. All kernel-mode drivers in 64-bit versions of Windows Vista and Windows 7 have to be signed by Microsoft [55].

The software could ship with its own measurement (e.g., hashes) signed by a trusted party. This would require some infrastructure, but would mean that the software itself could deliver the correct values, reliving the recipient of having to keep a huge database over correct values for different software versions.

Creating a foolproof chain of trust Everything has to be accounted for. The process has to include everything involved in any way with the boot process. If anything is forgotten, as seemingly happened with expansion cards in Bitlocker, unmeasured code can be executed. In addition, software is often complex and creating a foolproof chain of trust often relies on the integrity of all the software in the chain. Also, platform features and quirks used in unintended ways can break the security. The original Xbox is a good example of these problems, as it suffered from both software bugs and failures in hardware. The problem will get much worse with general computing, due to all the different configurations that has to be supported.

In 2007 Kauer found problems with all the three TPM enhanced bootloaders tested ([46]). They where all LILO or GRUB based, and suffered from faults in the implementation or faults in the design, such as loading files twice: once for checking and once for execution.

There is no easy fix for these problems, but standardisation and scrutiny of hardware, firmware and software will help.

When the attacker can pick and choose hardware and software, as in our case, one loophole in the whole system is enough.

Further changes in hardware and software are probably needed.

Extending the Chain of Trust Although it is often implied in TC literature that securing the boot process will fix all security problems, in reality, this is not enough. The chain of trust has to be extended beyond the

[13]More precisely a key pair. Being a non-migratable, the private key can not leave the TPM, "proving" that data signed with this key was signed internally in the TPM

boot process and into the operating system and programs, and it has to be maintained. This is done by always controlling which software runs on the platform. The security of an applications depends on the security of all software executed before it [56]. The reason for this is that the TPM is a passive chip and can not measure software on its own. Therefore the OS have to be extended with TPM support to either stop unwanted software or record what applications it runs. Although the application itself could measure (extend) itself into a PCR, this valued could not be trusted since the application itself is not trusted.

Even with OS support, a bug in one of the trusted programs (bound to happen) can break the chain of trust and allow for modification of that program or execution of unchecked and unmeasured code on the platform.

It can also be argued that todays operating systems and applications are so complex that stating facts about their security and integrity is almost impossible. This is one of the arguments used to push Dynamic Root of Trust technologies [56]. See section 4.4.

Time of check, time of use Measuring software before it is loaded means there is a time gap between the software check (found safe) and the software use. Software is also attested to at run-time, but was checked at load-time. Several critics point to this as a problem ([38]) since, e.g., a vulnerability in the software could be exploited, and the running software would no longer be secure. This is of course true, as the trusted boot itself does nothing to check software after it is loaded. But the real problem is actually to measure complex software. If the software is vulnerable it is not secure, should not be trusted, and in a secure boot, not loaded at all.

User Limitations If the user wants trusted or secure boot, he or she will be limited to approved hardware and software, meaning all kinds of custom, outdated or uncommon hardware and software will present problems.

There is a valid concern that going down this road will be exploited by certain vendors and lead to unwanted limitations on the user. See section 4.2.

Unintended use of legacy hardware and software Todays trusted and secure boot relies on common hardware and software that was not built with this in mind. Generally, it is not a good idea to rely on features for security if they where not built with security in mind.

In a way, we are "abusing" the boot process and legacy software for doing something it was not meant to do. All the use of legacy hardware does not

help this, as one has to have a complete understanding of all the quirks. An example of the problems this can lead to is described by Brossard. He found that pre-boot authentication often was implemented in a insecure way:

> Many pre-boot authentication software programmers are not aware of the inner workings of the BIOS interruptions they use in their products, which can lead them to wrongly assume the BIOS handles the keyboard in a secure way by itself. [57]

The first Xbox can be seen as a case in point for these difficulties. One advantage trusted boot has over the Xbox is that the goal is not to lock down the platform. It does not matter that the boot process can be hijacked as long as it is evident that the boot was not secure.

Lack of Memory Encryption Trusted boot and secure boot are only intended to measure and control what software can be executed. Thus trusted boot alone, without other TC technologies, can leave the system vulnerable to even "simple" hardware tampering techniques such as DMA attacks. DMA attacks abuse the fact that DMA[14] enabled devices can read from and write to the memory directly, independently of the CPU [58]. Also it might be possible to extract information directly from physical memory, either on a running system or shortly after it has been shut down [47]. This problem recently got more attention when the cold-boot attacks where proven practically possible [48].

Further protection such as memory encryption will often be necessary to keep the platform secure also after boot.

Privacy Concerns In a chain of trust all loaded during boot and henceforward play a part in the overall security of the system. Therefore, all software loaded need to be verified by a remote party for them to verify the trusted boot, even if they are only interested in one application. This, of course, can be a privacy concern. [56]

4.4 Dynamic Root of Trust Measurement

Since building a chain of trust from boot have proven to be challenging in practice, a few somewhat newer techniques have emerged. Dynamic root of trust measurements (DRTM) promises to bring the benefits of trusted boot without the hassle of controlling everything from the boot and forward. It is

[14] Direct memory access

a fundamentally different way of approaching the problem, and it does not give use the same platform control as a successful trusted boot, but is easier to implement.

Where static root of trust approaches, such as trusted boot and secure boot, build a chain of trust by measuring all code executed from the initial system start, dynamic root of trust approaches does not care about the boot process or what other code is running. Instead, these solutions strive to create a known-good state (root of trust) without requiring a reboot. In this known-good state, a hypervisor or other software is started. A dynamic root of trust can be created at any time after the system has booted, or during the late stages of the boot process such as in the boot loader.

This is rather impossible using legacy PC hardware as both running programs and previously executed code can impact the platform and interfere with the sensitive code. To create a known-good stat that is not affected by the current status of the platform, new hardware extensions beyond the TPM are needed. A dynamic root of trust is created using new CPU instructions, with support from other architecture changes and the TPM. Both AMD and Intel have created CPUs with support for dynamic root of trust, and these are are already availability in some off-the-shelf hardware [59]. AMD's solution is called Secure Virtual Machine (SVM) and Intel's solution is called Trusted Execution Technology (TXT). These technologies are different and not compatible, but for our purposes they can often be regarded as one because they both implement the same functionality [59]. They both offer hardware extensions such as new CPU instructions to create a dynamic root of trust (SKINIT and senter) and protection from DMA for selected portions of memory [46, 56].

Trusted Computing Base The collection of software that is trusted is sometimes referred to as Trusted Computing Base (TCB). In trusted boot and secure boot solutions this includes the BIOS, boot loader, operating system, drivers and other applications. Since it is believed that the number of bugs is proportional to the lines of source code in software ([60, 38]), minimizing he TCB is a way to limit the number of critical flaws in trusted code. Dynamic roots of trust excludes all other software from the TCB by breaking all ties to previously launched, as well as currently running, code. By not depending on any other software, we do not depend on their security properties either, and bugs in those programs will not affect the security of code in the TCB. But the TCB must be entirely self-sufficient and can not utilize other software such as common libraries or operating system functions. Code, if available from a secure location, can be included as a part of the

TCB, at the cost of expanding it. Keeping the TCB self-contained can be a challenge, but means that all this other code no longer have to be trusted.

4.4.1 The Open Secure Loader (OSLO)

After identifying several problems in the then-current static root of trust implementations (sections 4.3.3 and 4.5), Kauer goes on to describe a secure bootloader (OSLO) based on the AMD dynamic root of trust CPU instruction (SKINIT) [46]. The goal is to shorten the chain of trust and thus minimize the TCB. This is accomplished by creating a dynamic root of trust before executing the operating system. This removes the BIOS, firmware and other bootloaders from the TCB. However, this does nothing to solve the problem of measuring complex software. OSLO only guaranties the state of the platform at the point when the OS is launched. The operating system itself and all applications that are executed have to be measured and trusted. Thus, OSLO creates a trusted boot or secure boot similarly to the static root of trust approaches. But instead of beginning ("anchoring") this chain-of-trust at the first code executed at system startup (CRTM), a dynamic root of trust is created by the bootloader.

4.4.2 Flicker

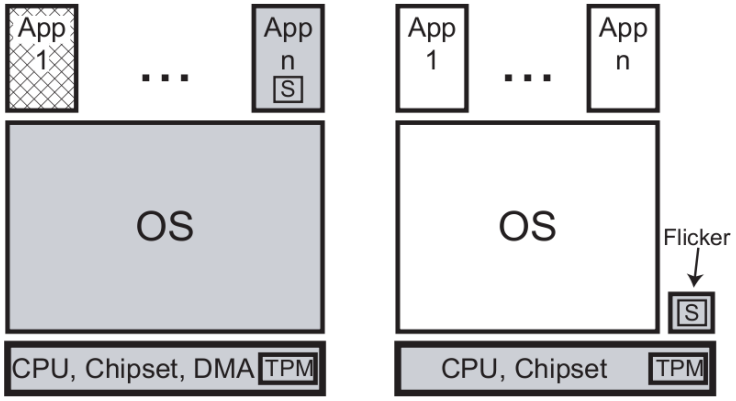

Figure 4.5: How code is executed under Flicker, from [56]. On the left, a traditional computer executing an application with a sensitive piece of code (S, or PAL). On the right, the application utilizes Flicker to run this sensitive code. The shaded areas represent the components that's trusted. On the left: Hardware, OS and applications with high privileges, and on the right: only Flicker itself and some of the hardware components.

Unlike many other DRTM solutions, Flicker ([61, 59, 56]) is not intended to launch an operation system (OSLO) or a hypervisor ([62], TrustVisor) securely; instead, it takes a different approach to the problem and focus on further minimizing the TCB. Flicker lets a developer run a piece of code contained and independent from other software on the system, including the BIOS, firmware, bootloaders, OS and applications. This piece of code, which they call Piece of Application Logic (PAL), is not intended to be an application. Rather the PAL should be a small, security sensitive part of an applications, for example, the part handling cryptography and secret keys. Every time the PAL is executed, Flicker builds a dynamic root of trust and suspends all other software. Thus, building a static chain-of-trust (4.3) is no longer necessary, and the TCB consist only of Flicker and the code to be executed Refer to figure 4.5. After the PAL has ran, Flicker will remove sensitive data from memory before the control is returned to the calling application.

Flicker also attests to a remote system that the code ran under Flicker and what the input and output of that code "block" (the PAL) was. As with all TC attestation, this must be done using the PCR values and TPM keys and certificates.. To secure the code block from the other software all ties to other code must be broken. On current hardware the OS and other applications must be suspended while the code block runs. This, obviously, impacts the performance of the whole system and can create other problems. Especially IO can be problematic if the code runs for too long. The PAL also has to be self-sufficient, it can not depend on any other code. Not only because it is the only code who actually runs, but more importantly, code used or otherwise depended upon is implicitly trusted. This means that the PAL can not, for example, make use of the standard Windows API or other libraries on the computer. Although standard code can be included in the PAL itself, this, in a way, defies the point since there is no reason to trust this code.

Flicker have some interesting properties. It relies on the TPM for measuring itself and the PAL's, and attesting these measurements to other systems. It also relies, as all DRTM solutions, on the CPU and chipset's ability to isolate the code executed under these new CPU instructions. Flicker requires a TPM and a CPU with the new security instructions, but no changes to the operating system or other software. The authors have created proof-of-concept code for Linux [59].

4.4.3 TrustVisor

TrustVisor ([60]) is a special-purpose hypervisor that represent the state of the art in both dynamic root of trust solutions and trusted computing[15].

Similarities to Flicker TrustVisor is inspired by Flicker (4.4.2, [59, 61]), and just as in Flicker, the goal is to be able to run security sensitive parts of an application (PAL) measured and isolated from other software. As with Flicker the TCB is kept small by not trusting the operation system or any applications except for TrustVisor itself (Fig. 4.6).

TrustVisor aims to have a much smaller performance overhead by executing most operations on the main CPU instead of in the TPM. Also, where Flicker have to create a new DRTM (SKINIT or senter instruction) for each invocation of a PAL, TrustVisor only creates one dynamic root of trust.

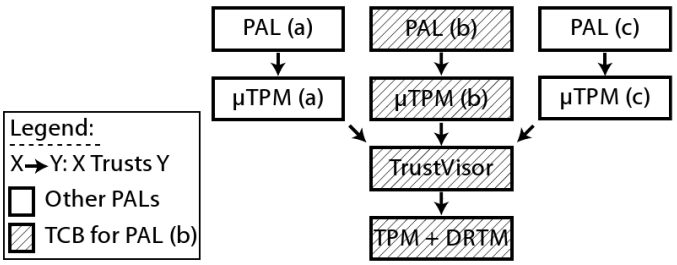

Figure 4.6: Trust in TrustVisor, from [60]. The security of the PAL depends on the TCB, which consists of the PAL itself, the μTPM and the rest of TrustVisor. It also depends on the hardware TPM and DRTM feature of the CPU and chipset (section 4.4), but not other hardware or software.

Launching TrustVisor

TrustVisor is a hypervisor and is started before the operating system. It runs with the highest privileges (ring 0) and can therefore control memory-mapping and hide memory from the OS and applications. Somewhat simplified, the bootloader will use DRTM (AMD's SVM in the reference implementation) to enable hardware memory protection, extend (measure) TrustVisor into a PCR and launch it. Thus, the DRTM operation creates a known-good starting point and the TPM will hold a hash of the TrustVisor code executed. The operating system (Linux in their implementation) is run as a

[15]The paper is scheduled to be presented at the IEEE Symposium on Security & Privacy in May 2010.

virtual machine. The goal of this is not to control or check the OS, but to have enough privileges to hide data from the OS and run code that it can not "tamper" with.

Running PALs

Registering Code is identified as a PAL when it is registered as such with TrustVisor through a application-level interface. What is registered is "a list of function entry points, and input and output parameter formats". Presumably, this means C code.

Invoking The functions that are part of the PAL can now be invoked as normal. When they are, the call traps to TrustVisor which sets up the secure execution environment. The PAL can only access its own memory. When the function is finished, TrustVisor again gets control and makes the function's return parameters available to the calling, untrusted, application. Also, the memory pages containing the PAL are marked as inaccessible.

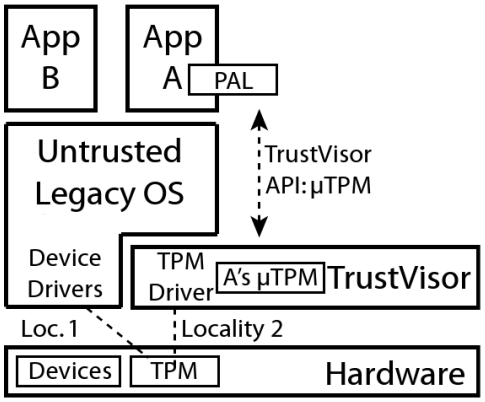

Figure 4.7: The PAL (and its functions) is registered with TrustVisor and executed in isolation from the operating system and other applications. The μTPM is the only interface exposed by TrustVisor to the PAL, but the real TPM is available for use if necessary.

Micro TPM Each PAL gets its own micro TPM (μTPM), a software-only implementation of some TPM features. The μTPM is a part of TrustVisor and therefore trusted. In order to keep the TCB small, the only TPM-like functions implemented are:

1. μPCRs. PCRs implemented in software.

2. Extend operation (HV_Extend) to measure code and data and store the result in the μPCRs

3. HV_GetRand for obtaining random data

4. Sealing and unsealing operations (HV_Seal and HV_unseal) for sealing data to μPCR values.

5. Quote operation (HV_Quote) to attest to the values of the μPCRs. This is done by signing the μPCR values with a key dedicated to this particular μTPM, which, in turn, is signed by a key in the real TPM.

These functions does not use the real TPM directly, but the main CPU. They therefore, presumably, run much faster[16].

Attestation and Trust Establishment

TrusVisor can attest to other systems that some code (the PAL) was run in an isolated execution environment. This is a two-step process: first TrustVisor must be attested to by the TPM, and then the PAL can be attested to by the μTPM.

Attesting TrustVisor The TPM signs the PCRs relevant for dynamic root of trust. This covers a measurement of TrustVisor itself, and the fact that DRTM was used correctly to launch it. The recipient must be familiar with TrustVisor to make sense of these values.

Attesting the PAL Before a PAL is executed it is extended into a μPCR. In adittion, the PAL can choose to extend other PCRs, e.g., with input and output parameters. With trust in TrustVisor the μTPM's HV_Quote operation can be used to sign these μPCR values and send them, together with the attestation from the real TPM, to the receiver.

4.5 General Problems and Challenges with Trusted Computing

These general problems and challenges with Trusted Computing technology. Problems specific to trusted boot and secure boot can be found in section

[16]The TPM is a low-cost chip and much less advanced than the CPU. The authors evaluation of TrustVisor confirms this [60].

4.3.3, and the challenges faced when implementing TC are discussed in section 4.6.

Reliance on Trusted Boot

Many TPM functions and TC use cases seem to, often implicit, require trusted boot. For most operations the TPM requires support in software such as the driver or the OS, which almost always means this software has to be trusted. In earlier (non-DRTM) literature trusted boot is seemingly taken for granted, as this get little focus in the trusted computing books and documentation. This is one example where the TC effort seems very little pragmatic and constantly looking too far into the future to be useful today.

Non-Compliant and Faulty TPM Implementations

In [46] Kauer discuss several problems found in TPM implementations. The first problem is a PCR reset attack against old (v. 1.1) TPM's. The second problem is faults in the then-current secure bootloaders (4.3.3). The last problem is the ability to overwrite the BIOS, including the CRTM, which is the static root of trust in the system. This is a violation of the specification which states the core root of trust must be at least somewhat protected against tampering. By overwriting the CRTM, an attacker can load a "hacked" bootloader and lie to the system by extending the correct PCR with the value of a correct bootloader.

In 2006 Sadeghi et al. tested many TPM implementations and found that several did not meet the specification [63]. This is worrisome because there is suppose to be a procedure in place for evaluation and certification of implementations [37, section 5].

Problems Related to Certificates

One area that seems largely forgotten, but still is crucial if we want to do anything useful with trusted computing, is the need for some sort of Public Key Infrastructure (PKI). Mentioned only briefly in TCG documentation (such as [37, section 6.4]), there might be a common consensus that developing PKI's are a separate and well known problem that will be solved elsewhere.

Much of what happens with the TPM depends on the ability of the receiver to check that incoming data was signed by an actual, correctly implemented, TPM. To do this, the recipient must have access to the certificate that vouched for the TPM's. A full-fledged PKI is probably not necessary, the problem can be solved by other means such as pre-shared keys and cer-

tificates. Indeed, this is how the similar problem is often solved when SSL is used, such as in web browsers.

The use of SHA-1

As Schneier discussed as early as 2005, SHA-1 is no longer considered a secure hash function, since collisions can be found quicker than a full brute-force attack [64]. In practice this is might not a problem for TPM use yet, as finding collisions is still very difficult. Unfortunately, these attacks have a habit of getting better with time.

Breakable Physical Security

In early 2010 Christopher Tarnovsky presented a talk[65] at the Blackhat conference in Washington, D.C. where he explained how he was able to break several Infinion security chips, among them their TPM. He stated that he used 6 months to accomplish this. His achievement was that he was able to extract secret keys from the devices that should never be accessible outside the device. Breaking one or more chips in this way does only break the security of those chips and does not affect the security of other chips or TPM in general, it is not a class break. It does, however, show how these chips resisted physical attacks, and it is reasonable to believe that once he has been able to break one, he will break identical chips faster in the future.

Huang believes that *"if you ship your secrets in your hardware, it is a good assumption that the users will eventually — and perhaps quickly — know your secrets"* [51, Section 4].

While the TPM only moves the problem, from attacking software to attacking hardware, this is a great improvement. Even though Hung is probably correct when stating that secrets in hardware are not totally secure, his own work on attacking the Xbox goes a long way towards proving the point that attacks on hardware are involved. Also, software attacks are often very easy to replicate. Even simple hardware attacks require some work.

4.6 Securing Massively Multiplayer Games with Trusted Computing

4.6.1 Using Trusted Boot and Secure Boot

The benefit of trusted boot and secure boot is the level of platform control it provides. Both Microsoft the Microsoft Xbox and Xbox 360 are examples of this, albeit the later a more successful one. Successful trusted boot solves many of the security problems currently seen in MMOs and other multiplayer games. Controlling or knowing what software is running on a system, would stop most of the known client-side game-hacking techniques. This is because the most common techniques involve either modifying or replacing the game-client, installed software and libraries or altering the operating system itself.

Trusted and secure boot is defined here as including the operating system, meaning that the it will continue the "chain-of-trust" started at boot. Arguably, this is outside the scope of the secure (or trusted) boot as it must be done by the OS, but the secure boot is a necessary prerequisite. Securing the boot process only, as Bitlocker currently does, it not enough to solve any problems. See section 4.3.3, "extending the chain of trust".

A modified attack graph is given in fig. 4.8. Motivation (starting points) are marked with gray, and problems solved with trusted boot and secure boot are grayed out. "Sell cheating programs" and problems related to password guessing are partially grayed out, since they will be severely hampered.

This section refers to the taxonomy in section 3.2.3.

Client-side Attacks Stopped

These attacks described in chapter 3 are either completely stopped or severely hampered by trusted and secure boot. Correctly working attestation would make software modification impossible, or rather, enable the server to tell modified clients from honest ones. Thus fixing the biggest security problem facing multiplayer games today.

Modifying the Game in Memory or on Disk (2bi, 2biiA, 2biiB) Section 3.5.2 describes the problem of game-client modification. With full control over the platform, the Xbox 360 does not allow any unknown games to run, and further stop modifications to the games and the operating system. Although the hardware is different, the same concept applies for trusted computing: with control over the platform, unwanted software can be detected or stopped from running. When the "chain-of-trust" is continued into the operating system, it will stop these attacks. Either, the OS can refuse to

load a modified game-client, or remote attestation will let the game server know that the client was modified. The server could refuse to serve clients that did not have a "proven" secure environment. But a better idea would probably be to treat them differently, by, for example, more actively monitor their actions in the game, or isolate these clients in separate shards.

Using the Layer Below (2biB, 2biiC, 2biiD) Section 3.5.3 defines this problem, which is similar to the previous one. The operating system itself is protected by the trusted or secure boot, and can therefore not be modified. The game-client itself can not be modified to load a custom driver. Run-time modification falls into the domain of OS security, but a trusted boot or a secure boot enables the OS to protect itself in a way not otherwise possible, because it can not be modified or replaced.

Automated playing (2b, 2c, 2d) Sections 3.5.1 and 3.5.4 define bots and automated play. Although controlling the interface can be done with hardware, it is more difficult than doing the same in software. Therefore "botting" and automated play typically require modifications to the game or operating system (DLL hooking). Since trusted boot boot and secure boot solves these problems, automated play becomes extremely difficult.

Automated play by abusing network traffic is covered later.

Other Attacks Stopped

Secure network traffic (3a) Since the platform and the software running can be trusted, it can also keep secrets. This enables encryption and integrity checking (MAC and public key signing) witch can fix these network related problems (3a), albeit with some overhead. Currently, keeping the key secure on the untrusted client is not possible. With secure storage the key can be sealed to the "valid" platform status. In addition, attestation could ensure that if "illegal" software was loaded to, e.g, dump the key from memory, the server would know and could discard this key.

Signing or encrypting the traffic is not enough to stop "botting" done by replaying network traffic. Fortunately, the game-client can now be trusted to include a nounce or timestamp. Also, generating valid network traffic from "scratch" is no longer possible beacuse it requires a valid key.

Help keep software and firmware up-to-date (1a, 1c, 2a) Vulnerabilities in software persists, but with trusted boot the servers can attest to each other, and to the client, that they where running the latest version of

the software. Similarly, the server can force the game-client to upgrade to the latest version. Although it is possible to check software versions without trusted boot, it relies on the software responding "truthfully".

Vulnerable hardware (or firmware) on the servers can also be detected.

Stop account thefts (1d, 1e) Authentication problems are discussed in 3.1.4. Using the unique identity of the TPM and the encryption included, a new authentication scheme could be devised that did not require the server to store usernames and passwords. Authentication could, for example, be a long shared secret the TPM on the client would release only when given the correct authentication info and in a secure state.

Also, since malware is not possible, and the network traffic can be encrypted, usernames and passwords are much more secure.

Other Benefits

Fewer checks on the server, more logic on the client The fact that the sever suddenly can start trusting the client, is a major gain. Today, developers are faced with the difficulty that most operations are most efficiently done at the client-side, but the game-client can not be trusted to keep secrets and do fair and impartial operations. Operations therefore often have to be executed at the server since it is the only trusted part, using precious server-side resources and adding lag to the equation. Developers are forced to try to strike a balance between security and performance. If the client can be trusted, more and more operations can be offloaded to the client, and the server can conserve resources today used to check client-side calculations, and otherwise test the clients integrity. Extending the client this way can also save bandwidth, which is a huge expense when running game servers. Also, when functions are moved to the client-side, waiting for the server is not necessary. This can increase the general performance of the game-client.

New features With trust in the client, new functions not possible today could be developed. Functions and features currently deemed too security sensitive to be on the client, but also too resource intensive to be implemented on the server, could now be implemented on the client. When the game-client is in a secure state (which the server can check) it can seal its own secret data to this state using the TPM. This data is only be accessible as long as the software has not been tampered with, stopping them from leaking.

Peer-to-peer gaming This also means that peer-to-peer gaming would become easier to implement. Today the lack of trust in other peers make it

difficult. Peer-to-peer MMOs would cut down the bandwidth requirements on the server-side even more, and spare the developers this great expense.

Remaining Issues

It is clear that trusted boot would not solve all the problems. Indeed, it only removes two top-level "branches" from the attack graph (4.8. This is, however, misleading. The attack graph is not weighted (section 3.8), but most security problems are related to client-side modification [12]. These are all solved with trusted boot.

Also, another major problem at the moment is account stealing. Current forms of account stealing use malware to either steal the account information or act as a man-in-the-middle. These problems are also solved with trusted boot.

Therefore I believe that trusted boot would solve the most serious problems.

Challenges

Unfortunately, there are many challenges that must be overcome before this can be used to secure multiplayer games. These are problems directly related to games, more general problems are given in section 4.3.3. Generally, trusted boot and secure boot can not be used to secure multiplayer games today, even when (or if) the hardware becomes widespread[17].

Lack of platform control Although the solutions taken by the console manufactures are interesting in their own right, they do not directly relate to computer gaming since the consoles are specialised systems controlled by the manufacturers to do one thing and one thing only. General computer systems are not controlled by the game developers and have to be flexible enough to be able to handle any task.

Trusted boot involve the whole system in a way that makes it very difficult for the game manufacturer alone implement. Both hardware and operating system support (section 4.3.3) are needed. It would be technically possible for the game developer, or more likely a third party, to develop their own "operating system" to use with games. This software could then support trusted boot or secure boot, and not be "at the mercy" of outside software updates. But, given the complexity of modern operating systems for general

[17] Although TPM modules are no longer rare, they are far from pervasive. The trusted computing group have believed for a long time that TPMs will "soon" be in every computer and mobile device. When, if ever, this will happen is impossible to predict.

purpose hardware, it it very unlikely that this would be a viable option. To make the matter worse, such an operating system would have to support the latest and greatest in 3D graphics and sound.

Also the OS have to take care not to run any software which can be exploited. On one hand, this seems almost impossible in a general purpose computing environment. Even if all needed software is pre-approved and signed, therefore trusted, one bug in one program can break the security. The chain of trust is just as secure as its weakest link. On the other hand, Microsoft do already have code signing for drivers, and Windows would not have to stop "unwanted" programs from running, which would put an even greater burden on the user, but only inform the TPM (by extending, or ruining, a PCR) that it has happened. One way to mitigate the problem is to take a lesson from the Xbox 360 design and use better privilege separation to stop the bug from affecting the whole system. It will fall into the hands of the OS to secure itself from the software it executes. The trusted or secure boot will keep the OS secure by checking it on startup, and then TPM features and other TC technologies can be utilized by the OS.

Therefore for trusted boot to be successfully used to secure games on the PC platform, this must be implemented by the operating system vendor, in most cases Microsoft, and the hardware manufacturers.

Fortunately many of the security challenge online games face are similar to more common software security challenges. Changes to the platform motivated by those problems will help secure games as well. The most promising happenings in this regard is Microsoft Bitlocker and the extended control they have implemented over drivers and kernel modules in the newer 64-bit versions of Windows. Although not enough, they might be a outlook of what is to come. Also, new CPU instructions have been added to both Intel and AMD CPUs, as discussed in 4.4.

Knowing the right answer Knowing the correct values to measurements is huge challenge (section 4.3.3). The approach used by Microsoft Bitlocker is to store the values at a given point in time, and later using these for reference. This approach is useless for our purpose because it makes it possible for the user of the platform to decide what the correct values are.

4.6.2 Using Dynamic Root of Trust

Dynamic root of trust is a more diverse technology than the static root of trust solutions trusted boot and secure boot. For our use, solutions that launch the operating system (OSLO) or a general purpose hypervisor are not useful. The reason for this is that they shorten the chain-of-trust, but

little else. This means that most of the problems described in sections 4.3.3 and 4.6.1 persist. OSLO can help launch the operating system securely, but the challenge of extending the chain-of-trust is the same. The problems when securing multiplayer games does not currently lie in the boot process.

Utilizing Flicker and TrustVisor

Flicker and TrustVisor can not deliver the same platform control trusted boot and secure boot can. Instead, these technologies can be regarded as building blocks for enhancing the security of applications. Therefore they must be built into the game-client and no general solution will fit all games. Even so, some observations and recommendations can be made.

Privilege separation As McCune et al. note in [60]:

> This design makes it the responsibility of application developers to identify the security-sensitive regions of their programs and group sets of functions into one or more PALs and untrusted portions. Essentially developers are required to perform privilege-separation

By separating the security sensitive portions of the game-client and running them under Flicker or TrustVisor, software modification (2b) would become less useful. What portions of the game-client, exactly, this is will vary from game to game. But generally, it will be code that handles the state of the gameworld and other secret information.

Similarly, automated playing using software (2c) becomes much more difficult if the game-client can protect the data structures typically read and changed when creating a bot.

Bugs in the software (1a, 2a) become less of a problem because a bug in the untrusted code can not give access to the PALs.

The major challenge in using Flicker will be to isolate code blocks (PAL's) that are both self-sufficient and small enough. Even with small PAL's, the performance impact might be too severe for resource intensive games. But this, of course, depends on the PAL and how often it has to run. In [59] McCune et al. develop a rootkit detector and find that executing it under Flicker takes approximately 1 second. The performance impact of using TrustVisor is much smaller, less than 7% in common cases, according to the authors ([60]). Unfortunately, some code that handles sensitive information, such as the gameworld state, is typically executed constantly.

Reference monitor Using TrustVisor it should be possible to develop a kind of reference monitor that would run and handle, e.g., all communication with the server. Such a "reference monitor" would keep a symmetric encryption key secret during runtime (the PALs memory pages can not be read from other software), and between runs using the sealing function of the μTPM. When the game-client would like to send or receive data from the server, it must go through this code to get it en- or decrypted. This would fix the network related problems (3a, 3d), except for DOS attacks.

The reference monitor could also be a client-side implementation of checks currently executed on the server, as a shared secret between the server and the reference monitor can make sure it can not be bypassed. This will save resources on the server-side and possibly enable more sophisticated checks.

This method could also possibly be used to enhance the authentication (1d,1e). Instead of just sending the username and password to the server for verification, a "black box" function could handle the authentication. It is not clear exactly how this would work. More research is needed.

Game-client checker Just as the rootkit detector McCune et al. develop using Flicker ([59]), a game-client "checker" could run infrequently, to measure the game-client and do other security checks. It would then be possible to attest to the server that this code actually ran, ran securely, and what the results were. The problem with this approach is that we again are using software to try to check or measure other software. It will be possible for, e.g., a modified game-client to "revert" back to normal behavior and hide itself before the secure code runs. McCune et al. argue convincingly that Flicker can guarantee, and remotely attest to, the execution of their rootkit detector. They do not, however, explain how a fool-proof rootkit detector would work. Supposedly, a rootkit (or game hack) could conceal itself before the PAL is ran, or hide in code not checked by the PAL. [61, 59]

Problems and Challenges

Some problems — other than the aforementioned runtime overhead and privilege separation challenges — surface.

Dynamic root of trust solutions do not require any modification of the operation system or platform, but they do require both a TPM and either AMD SVM or Intel TXT (section 4.4). Although processors motherboards with Intel TXT and AMD SVM are available off-the-shelf, they are far from pervasive. Hopefully, the relative ease of implementing some of these solutions, and backing from Intel and AMD, will mean that these technologies will become widespread. Time will tell.

Also, it is not clear how hard it is to attack the DRTM on the hardware level. DMA attacks are prevented, but how difficult is it to read the RAM using other methods? Although forcing hardware attacks, however simple, is an achievement; knowing how much protection these technologies provide is important.

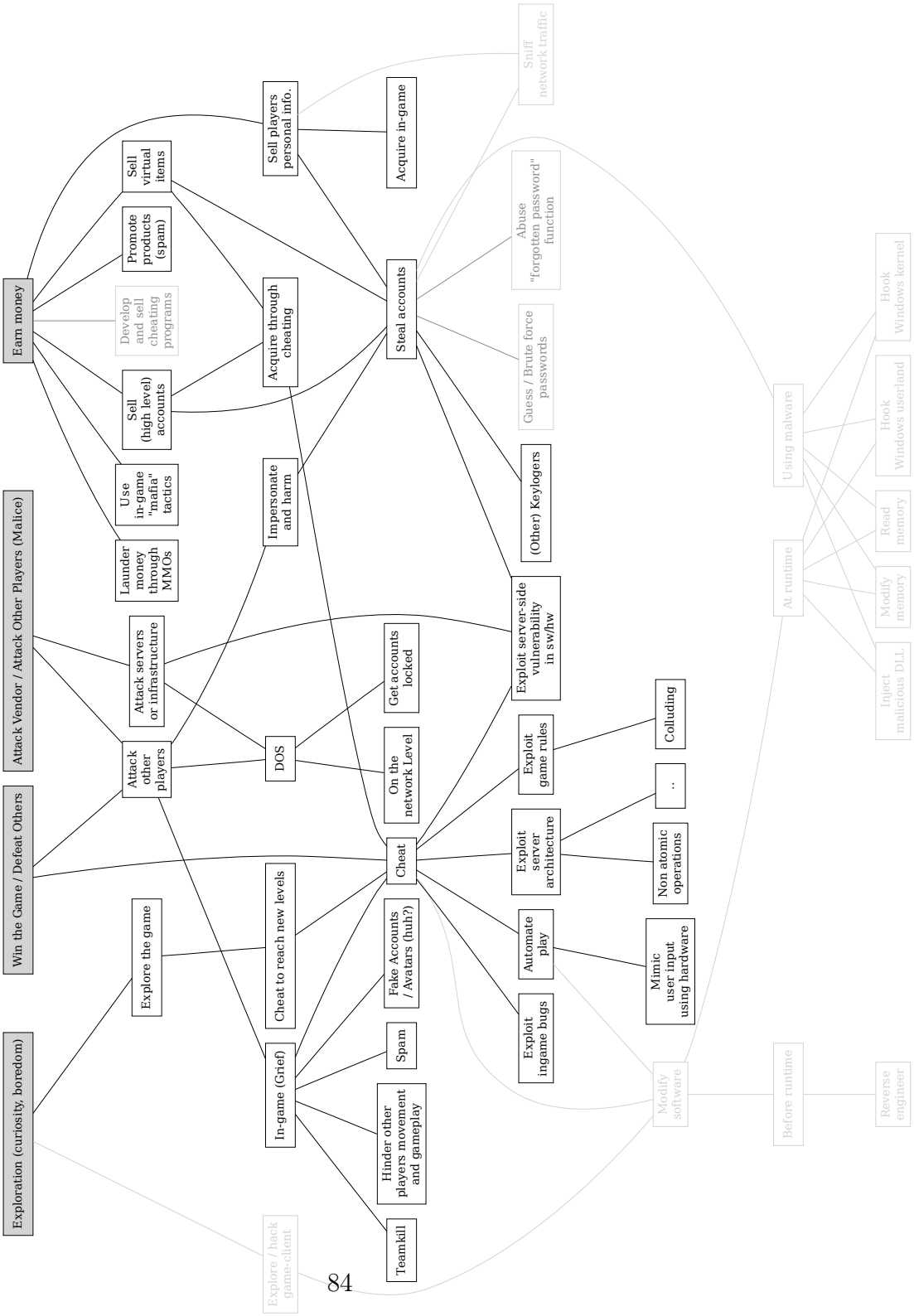

Figure 4.8: Modified version of 3.5.

Chapter 5

Conclusion and Future Work

The security problems related to Massively Multiplayer Online Games, and games in general, are many and diverse. This thesis have described these games and detailed their security problems. A state of the art overview of these problems based on current literature was given. A new taxonomy was developed, together with an attack-graph, to better understand the problems. Common motivational factors and attacks where identified and described. Trusted Computing technologies where researched, and how they can help solve the aforementioned problems where discussed.

Conclusion The classic trusted computing technologies, known as trusted boot and secure boot, would solve the major security issues. But due to the diversity of PC hardware and software, implementing this is a challenge yet to be completed. The game developers can not implement this alone, as it requires changes to the operating system and, possibly, hardware.

Re-designing the operating system most commonly used for gaming, Microsoft Windows, to fit this security model is no small feat, and can not be expected in the short-term. However, Microsoft, with Bitlocker, has taken some steps to homogenize PC hardware and implement trusted boot. Unfortunately, this does not extend into the operating system. Until this is implemented in Windows these technologies can be ignored.

The newer, dynamic root of trust based, technologies have great potential. Technologies such as Flicker and TrustVisor can be implemented in the game-clients without modifications to the operating system or hardware. This makes these technologies much more viable. Although they do not give the same level of control over the platform as trusted boot and secure boot would, they do ultimately have their root of trust in the hardware TPM. Therefore

they can attest to the trustworthiness of the system in a way not possible with software only solutions.

The hardware used, the TPM and CPU security extensions, are available but not, by far, in all gaming systems. They possibly will be, but promises of pervasive TPM chips have been made, and broken, many times since Trusted Computing was first purposed.

For any Trusted Computing technology to be useful, a few underlying problems must be solved. The most serious of which is probably the challenge of distributing certificates securely, as it is a prerequisite of all dynamic root of trust solutions.

Increased trust in the game-client would change the way Massively Multiplayer Online Games, and multiplayer games in general, are implemented, and would let the developers create features not possible today. This includes peer-to-peer gaming, which would be a major overhaul of the whole gaming architecture.

Future Work At the moment, TrustVisor looks like the most interesting solution. The project is still at the research stage, and further investigation into its properties and how it can be used is necessary. Although a prototype is implemented in Linux there is no Windows support for either Flicker or TrustVisor. Interesteing future work would be to port TrustVisor and Flicker to Microsoft Windows. Also, using TrustVisor to implement a trusted client in an open source game, such as Bzflag[1], could be challenging and interesting assignment.

[1] `http://bzflag.org/`

Bibliography

[1] World of warcraft subscriber base reaches 11.5 million worldwide (Blizzard Entertainment press release). Retrieved 20.04.2010 from http://us.blizzard.com/en-us/company/press/pressreleases.html?081121, November 2008.

[2] Nan Hu, Jianhui Huang, Ling Liu, Yingjiu Li, and Dan Ma. Wake up or fall asleep-value implication of trusted computing. *Inf. Technol. and Management*, 10(4):177–192, 2009.

[3] Shane Balfe and Anish Mohammed. Final fantasy - securing on-line gaming with trusted computing. In Bin Xiao, Laurence Yang, Jianhua Ma, Christian Muller-Schloer, and Yu Hua, editors, *Autonomic and Trusted Computing*, volume 4610 of *Lecture Notes in Computer Science*, pages 123–134. Springer Berlin / Heidelberg, 2007.

[4] Electronic Frontier Foundation. Trusted computing: Promise and risk. Retrieved 10.03.2010 from http://www.eff.org/wp/trusted-computing-promise-and-risk.

[5] Wikipedia: Trusted computing. Last retrieved 29.04.2010 from http://en.wikipedia.org/wiki/Trusted_Computing.

[6] The Entertainment Software Association. Essential facts about the computer and video game industry. Retrieved 17.04.2010 from http://www.theesa.com/facts/pdfs/ESA_EF_2009.pdf, 2010.

[7] Lu Fan. Solving key design issues for massively multiplayer online games on peer-to-peer architectures. Available at http://www.macs.hw.ac.uk/~lf16/thesis.pdf, May 2009.

[8] Jeffrey Kesselman. Server architectures for massively multiplayer online games. Available at http://barzilouik.free.fr/cnam/DEA_STIC_opt_CONCEPT_APP_MULTIMED_2005/UV_PetitOral/2_MMORPG_PbTech/ExposeOtherPeople/OV_MMORPG/Docs/

ts1351-SUNGameOne-ServerArchitecturesforMassivelyMultiplayerOnlineGames.pdf, 2004.

[9] Nathaniel E. Baughman, Marc Liberatore, and Brian Neil Levine. Cheat-proof playout for centralized and peer-to-peer gaming. *Networking, IEEE/ACM Transactions on*, 15(1):1–13, Feb. 2007.

[10] Global threat trends – year-end report 2009. Retrieved 22.04.2010 from http://www.eset.com/resources/threat-trends/EsetGlobalThreatReport(Jan2010).pdf, December 2009.

[11] Steven Davis. *Protecting Games: A Security Handbook for Game Developers and Publishers*. Charles River Media, Inc., Rockland, MA, USA, 2008.

[12] Greg Hoglund and Gary McGraw. *Exploiting Online Games*. Addison-Wesley, Boston, United States, first edition, July 2007.

[13] Battle.net authenticator faq. Retrieved 17.04.2010 from http://us.blizzard.com/support/article.xml?locale=en_US&articleId=24660.

[14] Eset annual global threat report: December 2008. Retrieved 29.07.2009 from http://www.eset.com/threat-center/threat_trends/EsetGlobalThreatReport(Jan2009).pdf, currently available at http://www.eset.com/resources/threat-trends/EsetGlobalThreatReport(Jan2009).pdf, December 2008.

[15] Wow authenticators bypassed by middlemen hackers - the register. Retrieved 17.04.2010 from http://www.theregister.co.uk/2010/03/02/warcraft_account_hack/.

[16] Mdy industries, llc v. Blizzard Entertainment, inc. et al. Available at http://docs.justia.com/cases/federal/district-courts/arizona/azdce/2:2006cv02555/322017/82/, July 2008.

[17] Fred von Lohmann. You bought it, you own it: Mdy v. Blizzard appealed. Retrieved 22.04.2010 from http://www.eff.org/deeplinks/2009/09/you-bought-it-you-own-it-mdy-v-blizzard-appealed.

[18] Wikipedia: Glider (bot). Retrieved 22.04.2010 from http://en.wikipedia.org/wiki/Glider_(bot).

[19] Justice has its price in sim world – The Boston Globe. Retrieved 29.07.2009 from http://www.boston.com/news/globe/living/articles/2004/01/14/justice_has_its_price_in_sim_world/.

[20] Øyvind Skaar. Organized crime in virtual worlds: How to get your own syndicate. In Audun Jøsang, Svein J. Knapskog, and Torleiv Maseng, editors, *Short-Paper Proceedings of the 14th Nordic Conference on Secure IT Systems (NordSec 2009)*, pages 30–36. UNIK, October 2009. Available at http://nordsec2009.unik.no/papers/Ska2009-NordSec.pdf.

[21] Kim-Kwang Choo and Russell Smith. Criminal exploitation of online systems by organised crime groups. *Asian Journal of Criminology*, 3(1):37–59, June 2008.

[22] j. yan and b. randell. An investigation of cheating in online games. *Security Privacy, IEEE*, 7(3):37 –44, may-june 2009.

[23] Jeff Jianxin Yan and Brian Randell. A systematic classification of cheating in online games. In *NETGAMES*, pages 1–9. ACM, 2005.

[24] Chris GauthierDickey, Daniel Zappala, Virginia Lo, and James Marr. Low latency and cheat-proof event ordering for peer-to-peer games. In *NOSSDAV '04: Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*, pages 134–139, New York, NY, USA, 2004. ACM.

[25] Steven Daniel Webb and Sieteng Soh. Cheating in networked computer games: a review. In *DIMEA '07: Proceedings of the 2nd international conference on Digital interactive media in entertainment and arts*, pages 105–112, New York, NY, USA, 2007. ACM.

[26] George Yee, Larry Korba, Ronggong Song, and Ying-Chieh Chen. Towards designing secure online games. In *AINA '06: Proceedings of the 20th International Conference on Advanced Information Networking and Applications*, pages 44–48, Washington, DC, USA, 2006. IEEE Computer Society.

[27] Matthew Pritchard. How to hurt the hackers: The scoop on internet cheating and how you can combat it. Retrieved 04.02.2010 from http://www.gamasutra.com/view/feature/3149/how_to_hurt_the_hackers_the_scoop_.php, July 2000.

[28] Jianxin Jeff Yan and Hyun-Jin Choi. Security issues in online games. *The Electronic Library*, 20(2):125–133, 2002.

[29] Knut Håkon T. Mørch. Cheating in online games - threats and solutions, version 1.0. Available from Norsk Regnesentral (NR, `http://www.nr.no`) upon request, January 2003.

[30] Wikipedia: Wallhacking. Retrieved 19.12.2008 from `http://en.wikipedia.org/wiki/Wallhacking`.

[31] Logan Lodge Bruce Potter. Fragging game servers. Available at `http://defcon.org/html/links/dc-archives/dc-17-archive.html`, 2009.

[32] Christian Mönch, Gisle Grimen, and Roger Midtstraum. Protecting online games against cheating. In *NetGames '06: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, page 20, New York, NY, USA, 2006. ACM.

[33] C. Collberg, J. Nagra, and W. Snavely. bianlian: Remote tamper-resistance with continuous replacement. Available at `ftp://ftp.cs.arizona.edu/reports/2008/TR08-03.pdf`, 2008.

[34] Stoogebot technical specs - a peek under the hood. Retrieved 04.03.2010 from `http://graphics.stanford.edu/~kekoa/stoogebot/tech.html`, 1997.

[35] Wikipedia: Exploit (online gaming). Retrieved 11.04.2010 from `http://en.wikipedia.org/wiki/Exploit_%28online_gaming%29`.

[36] C. Mitchell (Ed.). *Trusted Computing*. The Institution of Electrical Engineers, London, UK, November 2005.

[37] Tcg specification - architecture overview, revision 1.4. Available at `http://www.trustedcomputinggroup.org/files/resource_files/AC652DE1-1D09-3519-ADA026A0C05CFAC2/TCG_1_4_Architecture_Overview.pdf`, August 2007.

[38] David Challener, Kent Yoder, Ryan Catherman, David Safford, and Leendert Van Doorn. *A practical guide to trusted computing*. IBM Press, 2007.

[39] Tpm main specification part 1 - design principles, version 1.2, level 2 rev.103. Available at `http://www.trustedcomputinggroup.org/resources/tpm_main_specification`, July 2007.

[40] Siani Pearson. *Trusted Computing Platforms — TCPA technology in context*. Hewlett-Packard Company, Prestice Hall PTR, Upper Saddle River, NJ, USA, 2003.

[41] Michal Zalewski. *Silence on the Wire: A Field Guide to Passive Reconnaissance and Indirect Attacks*. No Starch Press, San Francisco, CA, USA, 2005.

[42] Richard Stallman. Can you trust your computer? Retrieved 10.03.2010 from `http://www.gnu.org/philosophy/can-you-trust.html`.

[43] Ross Anderson. "trusted computing" frequently asked questions - version 1.1. Retrieved 10.03.2010 from `http://www.cl.cam.ac.uk/~rja14/tcpa-faq.html`, August 2003.

[44] Lucky Green. Trusted computing platform alliance: The mother(board) of all big brothers. Available at `http://defcon.org/html/links/defcon-media-archives.html#DEF%20CON%2010`, August 2002.

[45] John Heasman. Implementing and detecting a pci rootkit. Available at `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.89.7305&rep=rep1&type=pdf`, November 2006.

[46] Bernhard Kauer. Oslo: Improving the security of trusted computing. In *Proceedings of the 16th USENIX Security Symposium (Online)*, pages 229–237, August 2007. Available at `http://www.usenix.org/event/sec07/tech/full_papers/kauer/`.

[47] Douglas MacIver. Pen testing windows vista bitlocker. Video available at `http://video.google.com/videoplay?docid=347395007131315532` and slides available at `http://conference.hackinthebox.org/hitbsecconf2006kl/materials/DAY%202%20-%20Douglas%20MacIver%20-%20Pentesting%20BitLocker.pdf`, September 2006.

[48] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM*, 52(5):91–98, 2009.

[49] Sven Türpe, Andreas Poller, Jan Steffan, Jan-Peter Stotz, and Jan Trukenmüller. Attacking the bitlocker boot process. In *Trust '09: Proceedings of the 2nd International Conference on Trusted Computing*, pages 183–196, Berlin, Heidelberg, 2009. Springer-Verlag.

[50] Michael Steil Felix Domke. "xbox" and "xbox 360" hacking - 17 mistakes microsoft made in the xbox security system & xbox 360 hacking. Available at http://events.ccc.de/congress/2005/fahrplan/events/559.en.html, December 2005.

[51] Andrew Huang. Keeping secrets in hardware: The microsoft xbox case study. In Christof Paar Burton S. Kaliski, Ãğetin K. KoÃğ, editor, *Cryptographic Hardware and Embedded Systems - CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 213–227. Springer Berlin / Heidelberg, 2003.

[52] Michael Steil Felix Domke. The xbox 360 security system and its weaknesses. Available at http://www.youtube.com/watch?v=uxjpmc8ZIxM, August 2008.

[53] Jeffrey Brown. Just like being there: Papers from the fall processor forum 2005: Application-customized cpu design. Retrieved 26.03.2010 from http://www.ibm.com/developerworks/power/library/pa-fpfxbox/, December 2005.

[54] Michael Steil Felix Domke. Why silicon-based security is still that hard: Deconstructing xbox 360 security. Available at http://events.ccc.de/congress/2007/Fahrplan/events/2279.en.html, December 2007.

[55] Security enhancements in Windows Vista. Retrieved 24.04.2010 from http://www.microsoft.com/downloads/details.aspx?FamilyId=6FB28358-68D9-43E9-B574-6A0D377BBA34&displaylang=en, May 2007.

[56] Jonathan M. McCune. Reducing the trusted computing base for applications on commodity systems. Available at http://sparrow.ece.cmu.edu/group/pub/mccunej_phd.pdf, January 2009.

[57] Jonathan Brossard. Bypassing pre-boot authentication passwords by instrumenting the bios keyboard buffer (practical low level attacks against x86 pre-boot authentication software). Availble at https://www.defcon.org/images/defcon-16/dc16-presentations/brossard/defcon-16-brossard-wp.pdf, 2008.

[58] Prashant Dewan, David Durham, Hormuzd Khosravi, Men Long, and Gayathri Nagabhushan. A hypervisor-based system for protecting software runtime memory and persistent storage. In *SpringSim '08: Proceedings of the 2008 Spring simulation multiconference*, pages 828–835,

San Diego, CA, USA, 2008. Society for Computer Simulation International.

[59] Jonathan M. McCune, Bryan J. Parno, Adrian Perrig, Michael K. Reiter, and Hiroshi Isozaki. Flicker: an execution infrastructure for tcb minimization. In *Eurosys '08: Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, pages 315–328, New York, NY, USA, 2008. ACM.

[60] Jonathan M. McCune, Yanlin Li, Ning Qu, Zongwei Zhou, Anupam Datta, Virgil Gligor, and Adrian Perrig. TrustVisor: Efficient TCB reduction and attestation. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2010. Accepted for IEEE Symposium on Security & Privacy, but not yet presented. Fetched from `http://www.ece.cmu.edu/~jmmccune/papers/MLQZDGP2010.pdf`.

[61] J.M. McCune, B. Parno, A. Perrig, M.K. Reiter, and A. Seshadri. Minimal tcb code execution. In *Security and Privacy, 2007. SP '07. IEEE Symposium on*, pages 267 –272, may 2007.

[62] Carl Gebhardt and Chris Dalton. Lala: a late launch application. In *STC '09: Proceedings of the 2009 ACM workshop on Scalable trusted computing*, pages 1–8, New York, NY, USA, 2009. ACM.

[63] Ahmad-Reza Sadeghi, Marcel Selhorst, Christian Stüble, Christian Wachsmann, and Marcel Winandy. Tcg inside?: a note on tpm specification compliance. In *STC '06: Proceedings of the first ACM workshop on Scalable trusted computing*, pages 47–56, New York, NY, USA, 2006. ACM.

[64] Schneier on security: Cryptanalysis of sha-1. Last retrieved 30.04.2010 from `http://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html`, February 2005.

[65] Christopher Tarnovsky. Hacking the smartcard chip. Available at `https://media.blackhat.com/bh-dc-10/video/Tarnovsky_Chris/BlackHat-DC-2010-Tarnovsky-DeconstructProcessor-video.m4v`, February 2010.